Découplage des services

L'**architecture monolithique** des applications traditionnelles cause plusieurs problèmes.

- Augmentation de la complexité du code
- Dépendance des différents modules
- Difficulté à modifier un module
- Propagation des erreurs
- L'arrêt d'un service arrête tous les services

Transition vers l'**architecture microservices**

- Chaque service est indépendant des autres
- Chaque service a sa responsabilité
- Si une erreur survient, elle va avoir une moins grande portée Affecte juste les services qui sont dépendants du service en panne
- Idéalement, chaque service serait indépendant des autres – Plus simple à modifier
- Moins de ligne de code pas service
- Peut faire une mise à jour d'un service indépendamment des autres Pour qu'une architecture microservice soit efficace, il faut aussi que son infrastructure soit découplée
- Chaque microservice à sa propre infrastructure • Gérer une multitude d'infrastructures qui s'intercommunique peut devenir rapidement chaotique
- On doit se soucier du couplage intraservice et interservices

Découplage synchrone

- Les distributeurs de charge permettent de découpler les instances web des instances app.
- Les instances d'app sont ajoutées au distributeur par le service d'auto-scaling
- Seulement les instances web connaissent le distributeur de charge
- Seulement le distributeur connaît les instances d'app
- Il faut au moins une instance app fonctionnelle pour qu'une requête provenant d'une instance web soit traitée, sinon la panne se répand aux instances web

Deux autres services de pour découpler nos services

- RabbitMQ
- Amazon SQS: service de files
- Amazon SNS: service de notifications

SNS

- D'avoir plusieurs senders / emiter
- Inscrit des messages dans la file
- D'avoir plusieurs receivers / worker
- Sonde activement la file SQS si des messages sont disponibles
- De transmettre des messages à n'importe quel débit
- De ne perdre aucun message
- De s'assurer que chaque message a été lu au moins une fois SQS ne garantit pas l'ordre de lecture des messages, par défaut
- Sauf si l'option FIFO est activée
- File de messages
- Répertoire pour les messages qui attendent d'être traités
- Agis en tant que tampon entre les composants qui génère les messages (senders) et les composants qui ingèrent les messages (receivers)
- Messages
- Peut contenir jusqu'à 256Ko de texte dans n'importe quel format
- Visibility timeout
- Permets de rendre un message invisible pendant un certain temps pour que le lecteur le traite
- Le lecteur après avoir traité le message, va le supprimer dans la file de SQS
- Si le lecteur tombe en panne, après l'expiration du délai le message redevient accessible dans la file
- Redondance des messages dans plusieurs serveurs
- L'ordonnancement des messages reçues
- Duplication des messages
- Fonctionnalité : Dead letter queue
- Permets de transférer des messages d'une file qui n'est pas capable d'être traitée
- Ex: Message erroné
- Après un certain nombre d'essaie (configurable) le message va être transféré dans un dead letter queue
- Permets aux responsables de diagnostiquer ce qui est arrivé avec le message
- Permets de corriger et retourner (replay) le message dans sa file pour être traité à nouveau
- Avantages
- Découple les receivers, des senders de façon asynchrone
- Permets de distribuer des messages à plusieurs receivers
- Si jamais tous les serveurs receivers sont en panne, le message est conservé dans la file d'attente
- Les messages sont traités lorsque les serveurs reviennent en ligne
- Les dead letter queue facilitent la gestion des messages erronés
- **Contexte**
- Un service de transcodage vidéo veut assurer une certaine qualité de service à leur client
- Par contre, chaque transcodage utilise à 100% le CPU pendant plusieurs minutes
- On veut ajouter des ressources en fonction du nombre de vidéos en attente de transcodage
- **Patron: Job Observer**
- **Objectif**
- Réagir à la quantité de tâches en attente pour ajuster les ressources utilisées
- **Avantages**
- Bon pour gérer le délai pour accomplir une tâche asynchrone en ajustant le nombre de tâches traité en parallèle
- Augmente la résistance aux pannes et aux fautes
- Une tâche échouée va retourner dans la file
- Permets d'optimiser l'utilisation des ressources
- **Problème**
- Les services de messagerie sont conçue pour transférer de petits messages
- Impossible de mettre des vidéos!
- **Patron: Claim-check**
- **Objectif**
- Décharger les données à traiter dans un espace de stockage externe et envoyer une référence vers le stockage externe dans le canal de communication
- **Avantages**
- Contourne la limite d'espace d'un message dans un canal de communication
- **Désavantage**
- Le client doit connaitre l'espace de stockage externe et le bus de communication
- Si le client tombe en panne avant d'envoyer le message dans le bus de communication, les données ne sont pas traitées.
- **Patron: Claim-check v2**
- Certains services de stockage cloud permettent l'utilisation d'événement
- Peut utiliser un événement pour envoyer un message dans le canal de message
- Avantages: Le client est découplé du canal de message, S'assure que tous les fichiers téléversés sont traités
- **Contexte**
- Nous avons un service Big Data qui permet à des clients de faire des rapports.
- Les clients peuvent s'abonner pour avoir leur rapport en moins de 5 minutes
- Les clients non abonnés n'ont aucune garantie de délai. Leurs rapports vont être créés selon la disponibilité du système.
- **Patron: File de priorité**
- Objectif: S'assurer que les tâches prioritaires sont exécuté rapidement tout en permettant l'exécution de tâches moins prioritaire
- **Comment**
- 1 file pour les tâches de basse priorité qui va être associée à un petit groupe de ressources (souvent fixe)
- 1 file pour les tâches prioritaires qui va être associée à un plus grand groupe de ressources
- Les deux groupes de ressources peuvent être ajustés séparément selon le nombre de tâches en attente
- **Avantage**
- Permets d'ajuster la performance pour accomplir les tâches prioritaires et non prioritaires individuellement
- Augmente la résistance aux pannes et aux fautes • Une tâche échouée va retourner dans la file
- Découple les serveurs qui tri les tâches par priorité et les serveurs qui exécutent les tâches

Désavantage

- Les serveurs qui tri les tâches doivent connaitre toutes les files de priorités
- **SNS: Simple Notification Service**
- Amazon SNS est un service qui permet de mettre en œuvre, opérer et envoyer des notifications à des applications externes qui lui sont abonnées
- **Concepts**
- **Sujets (topics)**
- L'entité où les clients/application s'abonnent pour recevoir les notifications
- **Abonnés**
- Représentent les différents clients/applications qui reçoivent les messages publiés dans un sujet
- Plusieurs types d'abonnés supportés – Courriel – SMS – HTTP/HTTPS GTI778 – Mobile push messaging
- Amazon SQS
- Message
- Texte jusqu'à 256Ko max dans n'importe quel format
- Un message est publié à un sujet Chaque message envoyé à un sujet correspond à (au moins) un message reçu par les abonnées
- Une fois un message publier à un sujet, il est impossible d'annuler l'envoi du message aux abonnées
- L'ordre de la réception des messages n'est pas garanti
- Des attributs peuvent être ajoutés pour caractériser le message – Chaque abonné peut définir un filtre sur les attributs pour recevoir que les messages désirés
- Permets de configurer un mécanisme de « retry » si l'appel de l'URL abonnée retourne une erreur. – Seulement pour les abonnés HTTP/HTTPS
- Persistance des messages
- Amazon SNS: Non
- Amazon SQS: Oui
- Mécanisme
- Amazon SNS: Push (Passif)
- Amazon SQS: Poll (Actif)

Producteur/Consommateur

- Amazon SNS: Publier/Abonner
- Amazon SQS: Envoyer/Recevoir
- Contexte
- On veut envoyer une tâche (message) à plusieurs files d'attente pour traiter une commande
- Quel est le danger de cette architecture?
- **Patron: Fan-out**
- **Objectif:**
- On veut envoyer la même tâche (message) à plusieurs files et on veut s'assurer que toutes les files le reçoivent.
- SNS est utilisé pour dédoubler la tâche (message) dans chaque file
- Chaque file va d'abord s'abonner au sujet SNS
- Chaque tâche reçue par SNS va être envoyée à toutes les files
- Chaque file va avoir un service distinct pour traiter les tâches reçues
- **Avantage**
- Les instances qui créent les tâches ont besoin de connaitre que le sujet SNS
- Pour ajouter ou enlever un module pour traiter une tache, les systèmes nécessitent que l'ajoute ou le retraie d'un abonnement au sujet SNS.
- Les instances des modules connaissent que leur file respective
- Les instances qui créent les tâches sont découplées des modules
- Cas d'utilisation: Fan-out - Vidéo
- **Patron: Event Bus**
- Architecture qui utilise un service de messagerie pour permettre les communications entre les différents modules du système
- Chaque module spécifie un filtre au service de messagerie lorsqu'il s'abonne au sujet SNS
- SNS va envoyer seulement les messages dont les attributs correspondent aux filtres des abonnés
- Chaque module peut envoyer un message (événement) aux services de messagerie
- Avantage
- Les modules n'ont pas besoin d'exposé d'interface particulière.
- Les modules vont s'entendre sur un lexique d'événement
- C'est le module qui définit quels événements qu'il peut gérer
- Chaque module est découplé des autres, toute communication passe par le bus d'événement
- Note: Très utilisé dans les architectures microservice serverless
- **Patron: File de priorité**
- Comment peut-on découpler les files de priorités des instances qui tri les tâches?
- **Comment**
- Avec l'utilisation de SNS, on peut découpler les instances qui tri les tâches des files de priorités
- Chaque file de priorité s'inscrit au sujet SNS avec un filtre correspondant à leur priorité
- Les instances qui tri les tâches publient un message dans le sujet SNS
- Chaque message doit avoir un attribut pour la priorité de la tâche
- SNS va envoyer le message à la bonne file de priorité
- Avantage
- Permets d'ajuster la performance pour accomplir les tâches selon leur priorité
- Augmente la résistance aux pannes et aux fautes
- Une tâche échouée va retourner dans la file
- Découple les serveurs qui tri les tâches par priorité et les files de priorité
- Désavantage
- Un message qui n'a pas l'attribut pour la priorité sera perdu.

- Les filtres ne peuvent pas détecter un attribut manquant
- SNS ne peut pas détecter un message qui a été envoyé à aucune cible
- Well-Architected framework
- Framework pour évaluer son infrastructure
- 5 piliers:
- Excellence opérationnelle
- Sécurité
- Fiabilité
- Efficacité vs efficience
- Optimisation des couts
- *Durabilité
- Donne des questions à se poser
- Ne donne pas de recettes toutes faites!

Haute disponibilité

- **Objectif**
- Minimiser le plus possible le temps d'arrêt d'un service sans nécessiter l'intervention humaine
- Éviter les points de défaillance unique
- **Facteurs**
- Tolérance aux fautes
- L'habilité d'un système à opérer malgré une panne
- Récupérabilité
- L'habilité d'un système à restaurer les données et services après une panne
- Extensibilité
- L'habilité d'un système à s'adapter à sa croissance future avec le moins d'effort possible.
- Récupérabilité du système
- Recovery time objective (RTO)
- Combien de temps le système peut prendre pour revenir à la normale
- Recovery point objective (RPO)
- Quelle quantité de données on est prêt à perdre lors d'une panne
- Si l'instance web ou app tombe en panne le service au complet tombe en panne
- Même si on démarre d'autres instances pour les remplacer, les adresses IP ne vont pas être automatiquement les mêmes.
- Floating IP: Il peut être attaché et détaché à une instance de façon dynamique
- Comment gérer une panne
- 1. L'auto scaling group détecte la panne et ferme l'instance
- 2. L'auto scaling group lance une nouvelle instance
- 3. Lors du démarrage de l'instance (User-data), le floating IP/interface est réassocié à la nouvelle instance
- 4. Le trafic est automatiquement redirigé à la bonne instance
- Avantages
- Très simple
- Peu ou pas de coût
- Vérifie l'état de l'instance
- Désavantages
- Le service est arrêté pendant tout le temps que la nouvelle instance démarre.
- Impossible d'ajouter des ressources pour traiter plus de requêtes.
- Ne vérifie pas la santé du service
- L'instance Web doit connaitre le IP de l'instance app
- Service de distribution de charge
- Permets de distribuer les requêtes à plusieurs instances
- Permets d'évaluer la santé des instances
- Comment gérer une panne
- 1. Le distributeur de charge détecte la panne du service et identifie l'instance comme étant « unhealty »
- 2. L'auto scaling group termine l'instance « unhealty »
- 3. L'auto scaling group démarre une instance et l'ajoute au taget group du distributeur de charge
- 4. Le distributeur attend que l'instance devienne « healty » avant d'envoyer des requêtes à la nouvelle instance
- **Avantages**
- Simple
- Vérifie l'état de l'instance et la santé du service
- Aucun temps d'arrête pendant
- Si au moins il y a deux instances par auto scaling group
- Possible de n'avoir aucune dégradation de service
- S'il y a suffisamment de ressources en redondance
- Facilite l'extension du service
- **Désavantages**
- Le service doit idéalement être stateless, sinon la fonctionnalité
- sticky session doit être utilisée.
- Ajoute des couts
- Des instances (selon la quantité de redondance désirée)
- 1 distributeur de charge
- Les distributeurs de charge et les auto scaling group peuvent s'étendre sur plusieurs zones de disponibilité
- Les communications interzone de disponibilité (inter-AZ) sont gratuites et sur des liens très haut débit
- Il n'y a aucune raison de ne pas faire de la redondance inter-AZ
- Partie 1:
- Pour assurer un service hautement disponible dans une seule région, il est nécessaire de déployer les composants minimaux suivants et de les assembler de manière appropriée :
- Load Balancer (équilibrage de charge) : Il répartit les requêtes entrantes sur plusieurs serveurs web pour distribuer la charge et assurer une disponibilité maximale.
- Groupe de serveurs web : Constitué d'au moins 2 serveurs web (pour la redondance) qui traitent les requêtes des clients. Les serveurs web doivent être configurés pour faire de la mise à l'échelle horizontale afin d'ajouter des serveurs supplémentaires en cas de besoin.
- Groupe de serveurs d'applications : Composé d'au moins 2 serveurs d'applications (pour la redondance) qui traitent les requêtes provenant des serveurs web. Les serveurs d'applications doivent également être configurés pour faire de la mise à l'échelle horizontale en cas de besoin.
- Base de données : Une base de données robuste et hautement disponible est nécessaire pour stocker et gérer les données de l'application. Vous pouvez utiliser un service de base de données géré avec réplication et sauvegardes régulières pour assurer la durabilité et la disponibilité des données.
- Le Load Balancer sera le point d'entrée pour les clients et dirigera les requêtes vers les serveurs web disponibles. Les serveurs web interagiront ensuite avec les serveurs d'applications pour traiter les requêtes, et ces derniers accéderont à la base de données pour récupérer ou modifier les données.
- Partie 2:
- Pour modifier l'infrastructure précédente afin de la rendre tolérante aux pannes, voici quelques suggestions :
- Utilisez un Load Balancer actif-passif ou un Load Balancer régional/global, qui basculera automatiquement vers un autre serveur web en cas de défaillance.
- Configurez les groupes de serveurs web et d'applications pour fonctionner en mode actif-passif ou actif-actif, en utilisant la réplication et la synchronisation des données entre les serveurs. Cela permettra de basculer rapidement vers un autre serveur en cas de défaillance.
- Mettez en place des mécanismes de surveillance et d'alerte pour détecter rapidement les problèmes de performance ou de disponibilité et pour automatiser la réponse aux incidents.
- Assurez-vous que la base de données est également tolérante aux pannes en utilisant des mécanismes tels que la réplication, les sauvegardes régulières et la bascule automatique vers un serveur de base de données secondaire en cas de défaillance du serveur principal.
- **Service de gestion de trafic via les DNS**

Fonctionnement

- 1. L'utilisateur tente d'accéder à « exemple.com » et son client DNS ne connaît pas l'adresse IP associée au nom de domaine
- 2. Le client DNS fait une requête de résolution du nom de domaine au serveur DNS
- 3. Le service DNS du fournisseur infonuagique, regarde si un record existe, celui-ci
- 4. Il applique la stratégie de routage pour choisir l'adresse IP à retourner
- 5. Il retourne l'adresse IP et le TTL (time to live, en sec) au client DNS
- 6. Le client DNS conserve la correspondance pendant la durée indiquée par le TTL
- Les prochaines requêtes seront résoutes directement par le client DNS
- 7. L'utilisateur communique directement avec le service via l'adresse IP
- Concepts
- **Nom de domaine (ou sous-domaine)**
- Nom de domaine vers le quelles le trafique est reçue
- **Health-check**
- Vérifie la santé des cibles via une requête HTTP

Stratégie de routage

- Méthode pour rediriger le trafic aux différentes cibles
- Différent d'un fournisseur à l'autre, mais plusieurs semblables
- **Cibles**
- Représente une ressource accessible publiquement ou à l'interne
- Adresse IP
- Nom de domaine (ou sous-domaine)
- Stratégie de routage
- **Priorité / failover**
- Toutes les cibles se voient associer une priorité
- Le trafic est toujours dirigé vers la cible en santé ayant le plus petit chiffre comme priorité
- Ex: Gérer la panne d'une base de données

Pondéré

- Toutes les cibles se voient associer un poids
- Le trafic est dirigé en fonction du poids de la cible
- $\%trafique = poid\ de\ la\ cible\ somme\ des\ poids\ des\ cibles\ en\ sant\acute{e}$
- Ex: Transition lors d'un changement de version d'une application

Stratégie de routage (suite)

- **Latence**
- Redirige le trafic vers la cible en santé dans la région qui répond le plus rapidement pour l'utilisateur final
- Le fournisseur garde et met à jour une table de latence par CIDR / Région
- Ne prends pas en compte la latence induite par votre service
- **Position géographique**
- Redirige le trafic selon la position géographique de l'utilisateur.
- Permet de redirigé le trafic provenant de certain continent, pays, états (US seulement) vers une cible.
- La provenance est déterminée selon l'adresse IP de la requête
- Ex: Permettre de garder toutes les informations personnelles des Canadiens en sol canadien.
- **Solution 3: Failover DNS Statique**
- Utilise le service DNS pour valider la santé du service dans une région
- Quand la région primaire tombe en panne, le service DNS redirige le trafic vers un site web statique hébergé par un service de stockage d'objet.
- Avantage
- Très simple
- Très peu de coûts
- Moins de 1$/mois pour le service de stockage d'objet
- Vérifier la santé de votre service
- Désavantage
- Le site web statique ne donne pas les fonctionnalités de votre service
- Le temps de basculement est régi par le TTL
- Certains ISP ont force un TTL minimum
- Note
- Souvent la solution utilisée pour les maintenances planifiées
- **Solution 4: Failover DNS Dynamique**
- Utilise le service DNS pour valider la santé du service dans une région
- Quand la région primaire tombe en panne, le service DNS redirige le trafic vers un service dans une autre région
- **Avantage**
- Vérifier la santé de votre service
- Permets d'avoir une redondance interrégion
- **Désavantage**
- Complexe à mettre en place
- Ajoute beaucoup de couts
- La synchronisation des données interrégion
- Varie selon la stratégie de récupération.
- Le temps de basculement est régi par le TTL
- Certains ISP force un TTL minimum
- **Cas spécial : Les bases de données SQL**
- Les bases de données ne supportent pas la mise à l'échelle horizontale. • Pour avoir des redondances multi-AZ ou interrégion, on doit utiliser d'autres stratégies • La fonctionnalité de synchronisation des serveurs SQL va permettre d'avoir un serveur SQL stand-by avec toutes les données
- Pour rediriger le trafic d'un serveur primaire à un serveur secondaire, nous pouvons utiliser le service de DNS pour faire un failover
- **Stratégie de récupération**
- Le but des stratégies de récupération de pannes commence où la haute disponibilité termine.
- Le plan de récupération détermine comment restaurer un système quand tout le système tombe en panne.
- Les stratégies de stratégies de récupération de panne sont choisies selon les objectifs RTO et RPO, en plus du prix pour la mise en place de celle-ci
- **Stratégie: Point de sauvegarde et restauration**
- Préparation
- Le système fait des points de sauvegarde périodiquement
- Les points de sauvegarde sont transférer vers un service de stockage d'objet

- Décrire la procédure de restauration du point de sauvegarde
- Image d'instance, user-data
- Comment mettre la nouvelle infrastructure en place
- Comment rediriger vers le nouveau système
- Dans le cas d'une panne majeure
- Restauration des points de sauvegarde depuis le service de stockage d'objet
- Déploiement de l'infrastructure nécessaire
- Rediriger vers le nouveau système
- **Avantages**
- Très simple
- Très peu de couts
- Facteurs:
- RTO: Heures: Temps requis pour
- Détecter la panne
- Déployer la nouvelle infrastructure (Minutes)
- Restaurer le point de sauvegarde (Heures)
- Rediriger vers le nouveau système – RPO:
- Temps depuis le dernier point de sauvegarde (souvent 24h)
- Couts
- Espace utilisée dans le service de stockage d'objet
- Transfert de données, si interrégion
- **Stratégie: Pilot light**
- Préparation
- Mettre en place l'infrastructure du système dans un autre endroit
- Image d'instances et user-data
- Plage réseau, sous-réseau
- Auto-scaling group (capacité min et désirée =0)
- Distributeurs de charge
- Seulement les instances nécessaires pour répliquer les données
- Aucune instance pour traiter des requêtes
- Répliquer les données dans une autre instance dans l'autre endroit
- Stratégie pour rediriger le trafic vers le nouveau système
- DNS failover*
- **Dans le cas d'une panne majeure**
- Rediriger vers le nouveau système
- Ajuster les quantités désirées pour gérer le trafic de production
- **Avantages**
- Peu de couts
- Seulement les distributeurs de charge et les serveurs requis pour la réplication.
- **Facteurs:**
- RTO: Temps requis pour
- Détecter la panne
- Réserver et démarrer les instances
- Dépends des stratégies de scaling
- Rediriger vers le nouveau système
- RPO: Dépends du type de réplications (souvent minutes ou moins)
- **Couts**
- Distributeurs de charge
- Instance de réplication
- Transfert de données, si interrégion
- **Stratégie: Warm stand-by**
- **Préparation**
- Même que Pilot-Light mais la capacité des auto-scaling group est un pourcentage du système de production
- Donc des instances pour traiter le trafic important sont toujours actives
- **Dans le cas d'une panne majeure**
- Rediriger le trafique vers le nouveau système
- Ajuster la quantité de ressource pour traiter le trafic du système de production
- **Avantages**
- Économique (vs répliquer tout le système de production)
- Peut traiter une partie du trafic de production à n'importe quel moment
- **Facteurs:**
- RTO:
- Pour le trafic important, le temps de rediriger le trafique
- Pour le reste, le temps de réserver les ressources nécessaires.
- RPO: Dépends du type de réplication (minutes ou moins)
- Couts
- Distributeurs de charge
- Proportion des instances utilisées pour traiter le trafic de production
- Transfert de données, si interrégion
- **Meilleures pratiques**
- Envoyer une partie du trafic pour tester que la stratégie fonctionne
- **Stratégie: Multi-site active-active**
- **Préparation**
- Comme Warm stand-by mais avec les capacités pour gérer tout le trafic de production
- Le trafic est traité par tous les sites
- Dans le cas d'une panne majeure
- Rediriger le trafique aux autres sites directement après la panne
- **Avantages**
- Peut traiter tout le trafique de production à n'importe quel moment
- Facteurs:
- RTO: Temps requis pour détecter la panne et rediriger le trafic
- RPO: Dépends de la réplication (minutes ou moins)
- Couts
- Double au minimum les couts
- Double d'instance
- Double de distributeur de charge
- Il faut déterminer les requis non fonctionnels de notre système
- Qualité de service
- Disponibilité
- Performance
- Déterminer le cout d'une panne
- Cout de temps d'arrêt
- Cout pour la perte de données
- Les couts augmentent lorsque
- La qualité du système augmente
- Les RPO et RTO diminuent
- Améliorer au fil du temps vos stratégies pour améliorer la qualité de votre système
- Tester vos stratégies de récupération!
- Ce n'est pas lors d'une panne majeure qu'on veut apprendre qu'elle ne marche pas
- Très courant en entreprise
- **Automatisation de l'infrastructure**
- Service de télémétries
- Surveille les différentes métriques des instances Lance des alarmes si certaines métriques dépassent une limite
- Service d'auto-scaling
- Réserve et lance de nouvelles instances
- Inscris les nouvelles instances au distributeur de charge (si applicable)
- Libère et ferme les instances sous-utilisées ou pas fonctionnelles
- Service de distribution de charge
- Distribue le trafic à toutes les instances inscrites qui sont en santé
- Vérifie la santé du service offert dans chaque instance
- **Service de distribution de charge**
- Le service de distribution de charge permet de réserver des distributeurs de charge à la demande
- Un distributeur de charge permet:
- De répartir des requêtes au sein d'un groupe d'instance selon leur charge de travail (envoie au moins achalandé).
- De distribuer les requêtes dans plusieurs zones de disponibilité de la même région.
- De vérifier la santé du service offert.
- D'avoir un seul point de contact
- 1 adresse IP publique (si public)
- Augmente la sécurité
- Distributeur couche 7 (AWS ALB, Octavia)
- Permets de distribuer des requêtes HTTP/HTTPS vers des groupes d'instances selon les informations contenues dans la requête. Ex: URL, Query
- **Concept:**
- Target group
- Groupe contenant tout les instances qui peuvent traiter les requêtes reçues par le distributeur de charge
- Chaque Target peut s'enregistrer avec son adresse IP (ou instance id) et un port
- Peut s'étendre sur plusieurs zones de disponibilité dans la même région
- Listener
- Entité qui reçoit les requêtes dans un certain protocole (HTTP/HTTPS), sur un certain port
- Rule
- Règle qui permet de rediriger une requête vers un Target group
- Ex: URL : « /images/* » : Target groupe 1
- Toujours au minimum une règle par défaut
- **Health-Check**
- Requête envoyer vers un URL pour évaluer la santé d'un service web dans un target group – Paramètres
- URL (HTTP ou HTTPS)
- Interval : Délai entre deux health-check
- Timeout : Délai d'expiration d'un health-check
- Healthy threshold count – Nombre de health-check réussi avant que le statut de l'instance soit healthy et reçois des requêtes
- Unhealthy threshold count
- Nombre de health-check échoué avant que le statut de l'instance soit unheathy et ne reçois plus de requêtes
- Fonctionnement 1. Une instance s'enregistre dans target group avec un état initial
- Le distributeur de charge n'envoie pas encore de requête à l'instance
- 2. Le distributeur de charge commence à faire ses health-check
- 3. Quand le nombre de health-check réussi (HTTP status 200) est au nombre du healthy threshold count l'instance change d'état à « healthy »
- 4. L'instance commence à recevoir des requêtes.
- 5. Quand le nombre de health-check échoué (HTTP status <> 200 ou délai expiré) est au nombre du unhealty threshold count l'instance change d'état à « unhealthy »
- 6. Le distributeur de charge n'envoie plus de requête à l'instance tant qu'elle ne redevient pas « healthy »
- **Certains services web sont stateful**
- Les requêtes d'un client doivent toujours être traitées par le même serveur.
- **Fonctionnalité : Sticky session**
- La fonction Sticky session utilise un cookie pour identifier un client
- Ce cookie est envoyé dans l'entête HTTP de chaque requête
- Chaque target group va associé le cookie avec une instance unique
- Danger des Sticky session
- La charge de travail est plus difficile à distribuer et risque d'être distribuée de façon non uniforme
- Si une instance est fermée, les informations sont perdues
- **Service de distribution de charges L7 •**
- Fonctionnalité: TLS offloading
- Permets aux distributeurs de charge de recevoir les requêtes en HTTPS et de les transférer aux instances en HTTP
- Seul le distributeur de charge nécessite d'avoir le certificat de sécurité pour TLS
- **Avantage**
- TLS a besoin d'un certificat sécurité et doit avoir une rotation de ceux-ci, donc lors de la rotation on met à jour que le distributeur et non toutes les instances.
- Crypter nécessite un traitement supplémentaire et n'ajoute aucun cout à l'opération d'un distributeur. !!!FREE CPU!!!
- Désavantage – Le flux d'information entre le distributeur de charge et l'instance n'est pas crypté
- Distributeur couche 4 (AWS NLB, LBAAS v2)
- Permet de distribuer des requêtes TCP / UDP / TLS
- Extrêmement rapide
- Impossible de rediriger le trafic selon la requête. • Seulement selon les ports
- 1 Règle (default) / Listener = 1 target group / Listener
- Permets d'utiliser le proxy protocol
- Permets d'avoir l'adresse IP de la source du message
- Permets d'utiliser le Sticky session
- Utilise l'adresse IP source pour lier à une instance
- !DANGER! : Très difficile à distribuer la charge quand plusieurs clients proviennent d'un même endroit (NAT)
- **Service de télémétrie**
- Objectif
- Centraliser tout les mesures provenant des ressources
- Analyser les mesures pour lever des alarmes de façon automatique
- Conserver l'historique des mesures
- Chaque instance à un agent du service de télémétrie
- Capture les mesures selon un intervalle fixe (ex: 1 minutes) et les publie aux service de télémétrie
- Exemple de métriques:
- % utilisation du CPU
- Quantité de bande passante utilisée
- Nombre d'erreur lancé par l'app. X
- **Alarmes**
- Permets de lancer une action lorsqu'une métrique dépasse une certaine limite pendant un certain nombre de temps pour une ressource ou un groupe de ressources.
- **Période (P)**
- Durée de temps analysé pour calculer un point selon une fonction (min, max, average, sum, count …) d'agrégation
- L'alarme est évaluée à la fin de chaque période **Point**
- Valeur utilisée pour évaluer une alarme
- **Nombre de points évalués (N) (Défaut 1)**
- Nombre de points évalués lors de l'évaluation de l'alarme
- **Nombre de points pour lancer l'alarme (M) (Défaut: 1)**
- Nombre de points qui dépasse la limite (supérieure ou inférieure) spécifiée par l'utilisateur pour lever l'alarme.
- **Limite**
- Valeur numérique

- Peut représenter une limite supérieure ou inférieure selon la fonction de comparaison
- **Fonction Comparaison (<, <=, >, >=)**
- Après l'évaluation de l'alarme, l'alarme peut être dans un de ces états:
- Ok
- Il y a eu moins de M points qui dépassent la limite pendant P*N • Alarme
- Il y a eu M ou + points qui dépassent la limite pendant P*N • Données insuffisantes:
- Il y a moins de N point disponibles pour évaluer l'alarme
- **Source**
- Source des métriques
- L'équivalent d'un Where dans une requête SQL
- **Actions**
- Appel d'un webhook
- Script d'automatisation d'infrastructure cloud – Ex: Auto-Scaling Policy (AWS)
- Fonction d'un service FAAS – Ex: Fonction en NodeJS dans OpenFaaS / AWS Lambda
- **Service d'auto-scaling**
- **Responsabilité**
- Réserver ou libérer des ressources sans intervention humaine pour avoir le nombre d'instances désiré en tout temps
- **Bénéfice**
- Utilisation efficace des ressources
- Amélioration de la disponibilité – Amélioration de la gestion des pannes
- Prendre pour acquis que TOUT peut tomber en panne!
- Diminution des coûts d'opération
- Payer seulement pour qu'est-ce que vous utilisez!
- **Stratégies de mise à l'échelle**
- **Verticale**
- Ajuster la capacité d'une ressource
- **Actions possibles :**
- Scale-down: Diminuer la capacité d'une ressource
- Scale-up: Augmenter la capacité d'une ressource
- **Horizontale**
- Ajuster la quantité de ressources utilisées
- Actions possibles:
- Scale-in: Libérer des ressources
- Scale-Out: Ajouter des ressources
- Objectif
- Faciliter la mise à l'échelle horizontale de vos services cloud.
- Concepts:
- Modèle de démarrage
- Contient toutes les informations pour démarrer une nouvelle instance dans un auto-scaling group.
- Id de l'image de base
- Type d'instance (Flavor) – Security group – Volume (service block storage) – User-data – Key pair
- OpenStack utilise un script HOT (service HEAT)
- Auto-Scaling group
- Groupe logique qui contient des instances semblables accomplissant la même tâche – Peut s'étendre sur plusieurs zones de disponibilité (AZ) d'une même région – Responsabilités
- S'assurer de contenir le même nombre d'instances fonctionnel que la capacité désirée
- S'il y a moins d'instances fonctionnelles que la capacité désirée, une instance est lancée selon le modèle de démarrage dans une des AZ spécifiées (AWS: Round-Robin)
- S'il y a plus d'instances fonctionnelles que la capacité désirée, une instance est fermée selon la stratégie configurée (AWS):
- Default (équilibre les AZ)
- Celle qui est le plus près de son heure complète d'exécution
- FIFO
- LIFO
- Auto-Scaling group
- Capacité désirée
- Nombre d'instances fonctionnelles désiré en tout temps
- Capacité minimale / maximale
- Capacité limite fixée par l'utilisateur que la capacité désirée ne peut jamais dépasser.
- Scaling Policy
- C'est l'élément qui ajuste la capacité désirée lorsqu'il est invoqué
- Action sur la capacité désirée
- {Add, Remove, Set to} # {% du groupe, # unité}
- Cooldown
- Temps n'attend avant d'effectuer une autre action d'autoscaling pour l'auto-scaling group
- Source d'invocation
- Alarme
- Événement planifié
- Distributeur de charge
- Un distributeur de charge peut être associé avec l'auto-scaling group en indiquant quel target group les instances doivent s'inscrire
- Permets aux nouvelles instances créées par le service d'autoscaling de recevoir des requêtes d'un distributeur de charge
- Health-Check – Permet d'utiliser le health-check pour terminer les instances non fonctionnelles – 2 types de health-check
- Service de machine virtuel
- Permets de détecter une instance qui ne réponde pas au système pour gérer le cloud (ex: bris matériel)
- Service de distributeur de charge
- Permets de détecter si le service est en santé
- Les bases de données SQL ne permettent pas la mise à l'échelle horizontale
- La mise à l'échelle verticale
- L'utilisation de Read-Replicas
- Le Sharding
- Infrastructure as Code
- Définition: Le processus de gérer et acquérir des ressources dans un centre de données en utilisant un fichier de définitions lisible par un ordinateur.
- Concept:
- Fichier de définition
- Json ou Yaml
- Gestionnaire de déploiement
- Terraform
- Gestionnaire spécifique au fournisseur cloud (AWS CloudFormation, Azure Automation, Google deployement manage, Heat)
- Stack
- Infrastructure générée par l'exécution du fichier de définition
- Répétabilité
- Déploiement de plusieurs stack depuis un fichier de définition
- Les fichiers de définition permettent de documenter l'infrastructure
- Si un répertoire git est utilisé pour les versionnés
- Augmente la traçabilité des changements
- Les fichiers de définition peuvent modifier un stack déjà existant, ce qui facilite la maintenance des stack
- La syntaxe des fichier de définition varie selon le gestionnaire de déploiement
- Les fichiers de définition de AWS CloudFormation peuvent être écrit en Json/Yaml
- Plusieurs sections sont présente dans les fichiers:
- Definition
- Matadata
- Rules
- Parameters
- Mappings
- Conditions
- Transform
- Ressources (Obligatoire)
- Outputs