

# LogAnalysis-5TB-Day

---

## Quick Start

**Nouveau?** Commencez par le **Executive Summary** (5 minutes de lecture) pour une vue d'ensemble rapide de la solution, des coûts et du ROI.

## Vue d'ensemble

Ce repository contient l'architecture et la documentation pour une solution d'analyse de **5 TB de logs par jour** en utilisant **Azure Data Explorer (ADX)** avec le langage de requête **KQL (Kusto Query Language)**.

## Contenu de la Documentation

### Executive Summary

#### **Executive Summary ★ START HERE**

Résumé exécutif de 5 minutes avec:

- Solution recommandée et justification
- Coûts résumés (4 scénarios)
- Architecture simplifiée
- ROI et bénéfices
- Comparaison alternatives
- Quick start en 30 minutes

### Architecture

#### **Architecture Globale**

Document principal décrivant l'architecture complète de la solution:

- Vue d'ensemble et objectifs
- Composants principaux (ADX, Event Hubs, Storage)
- Stratégies de déploiement et sécurité
- Haute disponibilité et disaster recovery
- Monitoring et optimisations

#### **Solution Technique ADX**

Détails techniques approfondis sur Azure Data Explorer:

- Pourquoi ADX pour l'analyse de logs
- Architecture détaillée du cluster
- Configuration et sizing (8-10 nœuds E16s\_v5)
- Schémas de tables optimisés
- Materialized views et update policies

- Best practices et cas d'usage réels

## ⌚ Ingestion de Données

### Stratégie d'Ingestion

Guide complet sur l'ingestion de 5TB/jour:

- **Option 1:** Azure Event Hubs (Streaming) - Recommandé
  - Latence < 2 minutes
  - Configuration 32+ partitions
- **Option 2:** Azure Blob Storage (Batch)
  - Coût réduit
  - Latence 5-15 minutes
- **Option 3:** Hybrid (Streaming + Batch)
- Transformation et enrichissement avec Update Policies
- Monitoring et troubleshooting

## 📊 Requêtes et Analyse

### Exemples KQL

Collection complète de requêtes KQL pour l'analyse de logs:

- Analyses de base (comptages, filtres)
- Analyse temporelle et time series
- Performance et latence
- Analyse d'erreurs et troubleshooting
- Sécurité et audit
- Distributed tracing multi-service
- Analytics business
- Monitoring et alerting
- Fonctions personnalisées et best practices

## 💰 Estimation Financière

### Estimation de Coûts

Analyse détaillée des coûts (Step 2):

- **Coût mensuel baseline:** \$15,000-20,000 (~\$180,000-240,000/an)
- **Coût optimisé avec RI:** \$13,000-15,000 (~\$156,000-180,000/an)
- Détail par composant (ADX, Event Hubs, Storage, Network)
- 3 scénarios: Production Optimisée, Startup, Enterprise HA
- Stratégies d'optimisation (Reserved Instances, Tiered Storage)
- Comparaison avec alternatives (Elasticsearch, Splunk, Datadog)
- ROI et justification business

## 🌐 Diagrammes d'Architecture

## Diagrammes PUML

Diagrammes PlantUML pour visualiser l'architecture:

### 1. [architecture-overview.puml](#)

- Vue d'ensemble complète de la solution
- Sources → Collection → Ingestion → ADX → Consommation
- Sécurité, monitoring, DR

### 2. [data-flow.puml](#)

- Flux de données détaillé end-to-end
- Séquence d'ingestion avec timings
- 5 étapes: Génération → Collection → Transport → Ingestion → Query

### 3. [adx-cluster.puml](#)

- Architecture interne du cluster ADX
- Engine nodes, storage tiers, data management
- Spécifications techniques par nœud

**Pour générer les images PNG depuis les fichiers PUML:**

```
# Installation PlantUML
brew install plantuml # macOS
# ou
sudo apt install plantuml # Ubuntu

# Génération des diagrammes
plantuml diagrams/*.puml
```

## ⌚ Solution Proposée

Architecture Recommandée

**Cluster Azure Data Explorer:**

- **8 nœuds** Standard\_E16s\_v5 (16 cores, 128GB RAM, 512GB SSD)
- **Capacité d'ingestion:** 1,600 GB/h (marge pour peaks 3x)
- **Hot cache:** 4 TB SSD (7-14 jours)
- **Hot storage:** 90 jours (~4.5 TB compressé)
- **Cold storage:** 1-2 ans (Azure Blob Cool Tier)

**Ingestion via Event Hubs:**

- **32 partitions** pour parallélisme
- **10-20 Throughput Units** avec auto-inflate
- **Latence:** < 2 minutes end-to-end
- **Format:** JSON compressé (GZip) ou Parquet

## Performance:

- **Requêtes:** Sub-seconde sur milliards d'événements
- **Compression:** Ratio 10:1 (5TB → 500GB stocké)
- **SLA:** 99.9% (99.99% avec multi-région)

## Métriques Clés

Métrique	Valeur
Volume quotidien	5 TB
Débit moyen	208 GB/h (~58 MB/s)
Débit peak (3x)	625 GB/h
Latence ingestion	< 2 minutes
Latence query (hot cache)	< 500ms
Compression ratio	10:1
Rétention hot	90 jours
Rétention cold	1-2 ans

## Coûts Estimés

Scénario	Mensuel	Annuel
<b>Baseline (sans optimisations)</b>	\$20,340	\$244,080
<b>Production Standard</b>	\$15,291	\$183,492
<b>Optimisé (RI 3 ans)</b>	\$13,465	\$161,580
<b>Startup Minimal</b>	\$5,000	\$60,000
<b>Enterprise HA Multi-région</b>	\$17,273	\$207,276

## 💡 Démarrage Rapide

### Prérequis

- Subscription Azure
- Azure CLI installé
- Permissions pour créer ressources (Contributor role minimum)

### Étape 1: Créer le Cluster ADX

```
# Variables
RG="rg-loganalysis-prod"
LOCATION="westeurope"
CLUSTER_NAME="loganalysis-prod-adx"
```

```
# Créer resource group
az group create --name $RG --location $LOCATION

# Créer cluster ADX
az kusto cluster create \
--cluster-name $CLUSTER_NAME \
--resource-group $RG \
--location $LOCATION \
--sku Standard_E16s_v5 \
--capacity 8 \
--enable-streaming-ingest true \
--zones "1,2,3"
```

## Étape 2: Créer Event Hub

```
# Event Hub Namespace
az eventhubs namespace create \
--name "loganalysis-prod-eh" \
--resource-group $RG \
--location $LOCATION \
--sku Standard \
--enable-auto-inflate true \
--maximum-throughput-units 20

# Event Hub
az eventhubs eventhub create \
--name "application-logs" \
--namespace-name "loganalysis-prod-eh" \
--partition-count 32 \
--message-retention 1
```

## Étape 3: Configurer ADX Database et Tables

Se référer à [adx-solution.md](#) section "Configuration Détailée" pour:

- Création de database
- Définition du schéma de table
- Ingestion mapping
- Data connection vers Event Hub

## Étape 4: Déployer Agents de Collecte

Se référer à [data-ingestion.md](#) section "Configuration des Agents" pour:

- Fluentd configuration
- Vector configuration
- Routage et buffering

## Ressources

### Documentation Officielle

- [Azure Data Explorer](#)
- [KQL Reference](#)
- [Event Hubs Documentation](#)

### Outils

- [ADX Web UI](#)
- [Azure Pricing Calculator](#)
- [KQL Playground](#)

### Tutoriels

- [ADX Best Practices](#)
- [Ingestion Best Practices](#)
- [Query Best Practices](#)

## Contributions

Ce repository est un document d'architecture. Pour des questions ou suggestions:

- Ouvrir une issue
- Proposer des améliorations via PR
- Contacter l'équipe Platform Engineering

## Licence

Documentation sous licence MIT - voir fichier LICENSE

## Checklist de Mise en Œuvre

### Phase 1: Foundation (Mois 1-2)

- Provisionner cluster ADX
- Configurer Event Hub ou Blob Storage
- Créer databases et tables
- Setup ingestion mappings
- Tester ingestion end-to-end
- Configurer monitoring de base

### Phase 2: Optimisation (Mois 3-4)

- Implémenter Update Policies
- Créer Materialized Views
- Optimiser hot cache policy
- Setup dashboards (Power BI/Grafana)
- Configurer alertes critiques

- Documenter runbooks

### Phase 3: Scaling (Mois 5-6)

- Acheter Reserved Instances
- Implémenter tiered storage
- Setup follower database (DR)
- Optimiser query performance
- Activer auto-scaling
- Audit de coûts

### Phase 4: Excellence (Mois 7+)

- ML pour anomaly detection
- Self-service analytics
- Advanced security (RLS)
- Multi-tenant isolation
- Continuous optimization

## Statut du Projet

### Step 1: Complété - Documentation Azure Data Explorer avec KQL

- Architecture globale
- Solution technique détaillée
- Exemples KQL complets
- Stratégie d'ingestion
- Diagrammes PUML

### Step 2: Complété - Estimation financière

- Coûts détaillés par composant
- 3 scénarios (Baseline, Optimisé, Enterprise)
- Comparaison avec alternatives
- Stratégies d'optimisation
- ROI et justification

---

**Version:** 1.0

**Date:** Juin 2024

**Auteur:** Architecture Team# Architecture Globale - Analyse de Logs 5TB/Jour

## Vue d'ensemble

Cette architecture décrit une solution complète pour l'analyse de **5 TB de logs par jour** en utilisant Azure Data Explorer (ADX), une plateforme hautement optimisée pour l'analyse de données massives en temps réel.

## Contexte et Objectifs

Contexte

- **Volume quotidien:** 5 TB de fichiers de logs
- **Besoin:** Analyse rapide, requêtes complexes, et visualisation
- **Échelle:** ~1.8 PB par an
- **Exigences:** Ingestion en temps quasi-réel, rétention flexible, performances élevées

## Objectifs de la Solution

1. **Performance:** Requêtes sub-secondes sur des milliards d'événements
2. **Scalabilité:** Supporter 5TB/jour avec possibilité d'évolution
3. **Disponibilité:** SLA 99.9% minimum
4. **Coût-efficacité:** Optimisation des coûts de stockage et compute
5. **Facilité d'utilisation:** Interface KQL intuitive pour les analystes

## Composants Principaux

### 1. Azure Data Explorer (ADX) - Cœur de la Solution

#### Caractéristiques clés:

- Moteur de requête distribué optimisé pour l'analyse de logs
- Compression native (~10:1 en moyenne)
- Indexation automatique de toutes les colonnes
- Support natif du format JSON, CSV, Parquet, Avro

#### Configuration recommandée:

- **Cluster:** Type Engine Standard\_E16s\_v5+
- **Nombre de nœuds:** 6-10 nœuds (évolutif selon charge)
- **Capacité d'ingestion:** ~200-250 GB/heure/nœud
- **Cache SSD:** Données chaudes (7-30 jours)
- **Stockage froid:** Azure Blob Storage pour archives

### 2. Couche d'Ingestion

#### Options d'ingestion:

##### Option A: Azure Event Hubs (Recommandé pour streaming)

- **Débit:** Jusqu'à 1 GB/sec par partition
- **Partitions:** 32 partitions minimum pour 5TB/jour
- **Rétention:** 1-7 jours dans Event Hub
- **Avantages:** Faible latence, découplage, buffer automatique

##### Option B: Azure Storage (Batch)

- **Pattern:** Drop de fichiers dans Blob Storage
- **Event Grid:** Notification automatique vers ADX
- **Format:** Compressed JSON, Parquet recommandé
- **Avantages:** Simple, économique pour batch

## Option C: Hybrid

- Streaming pour logs critiques via Event Hubs
- Batch pour logs historiques via Storage

## 3. Transformation et Enrichissement

### Update Policy dans ADX:

```
// Politique de mise à jour pour enrichissement automatique
.create-or-alter function EnrichLogs() {
    RawLogs
    | extend
        ParsedTimestamp = todatetime(timestamp),
        Severity = case(
            level == "error", "High",
            level == "warning", "Medium",
            "Low"
        ),
        Region = extract(@"region=([^\,]+)", 1, metadata)
    | project-away raw_field
}
```

## 4. Stockage Hiérarchisé

### Stratégie de rétention:

- **Hot Cache (SSD)**: 7-14 jours - Requêtes ultra-rapides
- **Hot Storage**: 90 jours - Performance optimale
- **Cold Storage**: 1-2 ans - Coût réduit, accès occasionnel
- **Archive**: > 2 ans - Export vers Azure Archive Storage

## 5. Sécurité et Conformité

### Contrôles d'accès:

- **Azure AD Integration**: Authentification centralisée
- **Row Level Security (RLS)**: Isolation par tenant/client
- **Chiffrement**: At-rest (Azure Storage Encryption) et in-transit (TLS 1.2+)
- **Audit**: Logs d'audit ADX activés

### Compliance:

- GDPR: Support PII masking et data retention policies
- SOC 2, ISO 27001 compliance native Azure

## Architecture de Déploiement

### Configuration Multi-régions

## Région primaire (ex: West Europe):

- Cluster ADX primaire
- Event Hubs namespace primaire
- Compute pour ingestion

## Région secondaire (ex: North Europe):

- Cluster ADX en standby ou lecture
- RéPLICATION via Follower Database
- Disaster Recovery: RPO < 1h, RTO < 4h

## Réseau et Connectivité

- **Virtual Network:** Intégration VNet pour isolation
- **Private Endpoints:** Connexions privées pour ADX
- **Service Endpoints:** Event Hubs, Storage sécurisés
- **Firewall:** IP whitelisting sur cluster ADX

## Flux de Données

1. **Génération de logs** → Applications/Services
2. **Collection** → Agents (Fluentd, Logstash, Azure Monitor Agent)
3. **Streaming** → Event Hubs (buffering)
4. **Ingestion** → ADX (compression, indexation)
5. **Transformation** → Update policies (enrichissement)
6. **Stockage** → Tiered storage (hot → warm → cold)
7. **Analyse** → KQL queries via portails/APIs
8. **Visualisation** → Power BI, Grafana, Azure Dashboards

## Patterns d'Usage

### Use Case 1: Monitoring Temps Réel

- Dashboards en temps réel (rafraîchissement 30s)
- Alertes automatiques sur anomalies
- Latence: < 2 minutes end-to-end

### Use Case 2: Investigations de Sécurité

- Requêtes ad-hoc sur 90 jours de données
- Corrélation d'événements complexes
- Time to insight: Secondes à minutes

### Use Case 3: Analytics Historiques

- Tendances mensuelles/annuelles
- Capacité planning
- Requêtes sur cold storage: Minutes

# Intégrations

## Sources de Logs

- **Azure:** Application Insights, Azure Monitor, Activity Logs
- **On-premises:** Syslog, Windows Events via agents
- **Applications:** Logs applicatifs JSON/structurés
- **Sécurité:** Firewall logs, IDS/IPS, WAF

## Consommateurs

- **Power BI:** Dashboards exécutifs
- **Grafana:** Dashboards opérationnels
- **Azure Logic Apps:** Automatisation basée sur requêtes
- **APIs REST:** Intégration applications custom
- **ADX Web UI:** Exploration interactive

# Évolutivité

## Scaling Vertical

- Augmentation SKU des nœuds (E8 → E16 → E32 → E64)
- Impact: Plus de RAM, CPU pour requêtes complexes

## Scaling Horizontal

- Ajout de nœuds au cluster
- Impact: Plus de débit d'ingestion et parallélisme des requêtes

## Auto-scaling

- Configuration basée sur:
  - CPU utilization (> 70%)
  - Ingestion queue depth
  - Query latency metrics

# Monitoring et Observabilité

## Métriques clés

- **Ingestion:**
  - Events ingested/sec
  - Ingestion latency
  - Failed ingestions
- **Query Performance:**
  - Query duration (p50, p95, p99)
  - Queries per second
  - Cache hit ratio

- **Ressources:**

- CPU/Memory utilization
- Disk IOPS
- Network throughput

## Outils de Monitoring

- **Azure Monitor:** Métriques et alertes
- **ADX Insights:** Dashboard intégré
- **Log Analytics:** Meta-monitoring des logs ADX
- **Application Insights:** Pour applications connectées

## Haute Disponibilité

### Résilience du Cluster

- **Nœuds multiples:** Minimum 3 nœuds pour HA
- **RéPLICATION:** Data repliquée sur multiple nœuds
- **Zone Redundancy:** Distribution sur Availability Zones

### Disaster Recovery

- **Follower Databases:** Lecture seule en région secondaire
- **Backup:** Continuous export vers Storage
- **Recovery:** Restore from Storage ou promote follower

## Optimisations

### Performance

1. **Partitioning:** Par date (colonnes datetime)
2. **Materialized Views:** Pré-agrégations pour requêtes fréquentes
3. **Cache Policy:** Optimisation du hot cache
4. **Extent Management:** Merge policy pour petits extents

### Coûts

1. **Tiered Storage:** Données froides en Azure Storage
2. **Reserved Capacity:** Réservations 1-3 ans
3. **Continuous Export:** Archive vers Blob Storage
4. **Auto-pause:** Dev/Test clusters en dehors heures

## Conformité et Gouvernance

### Data Governance

- **Data Classification:** Tags sur tables/colonnes
- **Retention Policies:** Automatiques par table
- **Data Lineage:** Tracking via metadata

## Audit et Logging

- **Audit Logs:** Toutes opérations DDL/DML
- **Query Audit:** Tracking des queries utilisateurs
- **Access Logs:** Connexions et authentifications

## Roadmap et Évolution

### Phase 1 (Mois 1-2): Foundation

- Déploiement cluster ADX de base
- Ingestion batch via Storage
- Requêtes KQL basiques

### Phase 2 (Mois 3-4): Optimisation

- Migration vers streaming Event Hubs
- Update policies pour enrichissement
- Dashboards Power BI/Grafana

### Phase 3 (Mois 5-6): Scaling

- Multi-region setup
- Optimisations performance avancées
- Intégrations additionnelles

### Phase 4 (Mois 6+): Excellence Opérationnelle

- Auto-scaling automation
- ML pour détection anomalies
- Self-service analytics pour équipes

## Références

- [Azure Data Explorer Documentation](#)
- [KQL Quick Reference](#)
- [ADX Best Practices](#)
- [Pricing Calculator](#)

## Documents Connexes

- [Solution Technique ADX](#)
- [Exemples KQL](#)
- [Stratégie d'Ingestion](#)
- [Estimation de Coûts](#)
- [Diagrammes d'Architecture](#)

## Estimation Financière - Solution Azure Data Explorer 5TB/Jour

## Résumé Exécutif

Cette estimation financière détaille les coûts mensuels et annuels d'une solution Azure Data Explorer pour l'analyse de **5 TB de logs par jour**.

### Coûts Mensuels Estimés

Composant	Coût Mensuel (USD)	% du Total
<b>Azure Data Explorer Cluster</b>	\$13,824	68%
<b>Event Hubs</b>	\$3,456	17%
<b>Stockage (Hot + Cold)</b>	\$2,160	11%
<b>Network Egress</b>	\$500	2%
<b>Monitoring &amp; Management</b>	\$400	2%
<b>TOTAL</b>	<b>\$20,340</b>	<b>100%</b>

Coûts Annuels: **~\$244,080**

## Détail par Composant

### 1. Azure Data Explorer (ADX) - Cluster Principal

#### Configuration Recommandée

##### Cluster Compute:

- **SKU:** Standard\_E16s\_v5
  - vCPUs: 16 cores
  - RAM: 128 GB
  - SSD Cache: 512 GB
- **Nombre de nœuds:** 8 nœuds
- **Région:** West Europe (exemple)

#### Calcul des Coûts ADX

**Prix unitaire** (West Europe, tarifs juin 2024):

- Standard\_E16s\_v5: ~\$1.44/heure/nœud

##### Coût mensuel:

$$8 \text{ nœuds} \times \$1.44/\text{h} \times 730\text{h}/\text{mois} = \$8,409.60/\text{mois}$$

**Markup Azure:** ~20% pour gestion et overhead = **\$10,091/mois**

## Options de Réduction des Coûts

### Reserved Instances (1 an):

- Réduction: ~38%
- Nouveau coût: **\$6,256/mois** (~\$75,000/an)
- Économie: **\$3,835/mois** (\$46,000/an)

### Reserved Instances (3 ans):

- Réduction: ~62%
- Nouveau coût: **\$3,835/mois** (~\$46,000/an)
- Économie: **\$6,256/mois** (\$75,000/an)

## Dimensionnement Alternatif

Configuration	Nœuds	SKU	Coût/mois	Use Case
<b>Minimal</b>	6	E8s_v5	\$4,200	Dev/Test
<b>Standard</b>	8	E16s_v5	\$10,091	Production (baseline)
<b>Optimisé</b>	8	E16s_v5 + RI 3ans	\$3,835	Production (optimisé)
<b>Premium</b>	10	E16s_v5	\$12,614	Peak loads, HA
<b>Enterprise</b>	12	E32s_v5	\$34,560	High concurrency

**Recommandation:** Commencer avec configuration Standard, puis optimiser avec Reserved Instances après 3-6 mois.

## Coût Stockage ADX (Hot Storage)

### Hot Storage (90 jours de rétention):

- Volume brut: 5 TB/jour × 90 jours = 450 TB
- Après compression 10:1: 45 TB
- Prix: ~\$0.08/GB/mois
- Coût mensuel: 45,000 GB × \$0.08 = **\$3,600/mois**

### Hot Cache (SSD, inclus dans compute):

- Capacité: 8 nœuds × 512 GB = 4 TB
- Rétention: 7-14 jours
- Coût: **Inclus dans le coût des nœuds**

**Total ADX (Compute + Hot Storage): \$13,691/mois** (sans RI)

## 2. Azure Event Hubs

### Configuration Recommandée

## Namespace:

- **Tier:** Standard
- **Throughput Units (TU):** 10 TU base, auto-inflate jusqu'à 20 TU
- **Partitions:** 32 partitions
- **Retention:** 1 jour

## Calcul des Coûts Event Hub

### Coûts de base:

- Base namespace (Standard): ~\$10/mois
- Throughput Units: \$30/TU/mois
- Average TUs utilisés: ~10 TU
- Ingress events:  $5 \text{ TB/jour} \div 500 \text{ bytes/event} = \sim 10 \text{ milliards events/jour}$

### Coûts mensuels:

```

Base: $10/mois
Throughput:  $10 \text{ TU} \times \$30 = \$300/\text{mois}$ 
Ingress events:  $10\text{B events/jour} \times 30 \text{ jours} = 300\text{B events/mois}$ 
 $300\text{B events} \div 1\text{M} = 300,000 \text{ million events}$ 
Premier 100M: Gratuit
200M-6000M:  $\$0.028/\text{million} = 5,900 \times \$0.028 = \$165.20$ 
Restant:  $294,100\text{M} \times \$0.011 = \$3,235.10$ 

Total Event Hub: ~$3,710/mois

```

**Estimation simplifiée: \$3,456/mois** (avec optimisations)

## Options Alternatives

### Premium Tier:

- Processing Units (PU): 1 PU
- Coût: ~\$672/PU/mois
- Avantages: VNet isolation, plus de capacité
- Recommandé si: Besoin isolation réseau forte

### Dedicated Cluster:

- Capacity Units (CU): 1 CU
- Coût: ~\$8,760/mois
- Avantages: Dédié, prévisible, pas de throttling
- Recommandé si: > 10 TB/jour ou très strict SLA

## 3. Azure Blob Storage

## Configuration pour Stockage Froid

### Cold Storage (1-2 ans au-delà de hot):

- Volume:  $5 \text{ TB/jour} \times 365 \text{ jours} = 1,825 \text{ TB/an}$
- Après compression 10:1:  $182.5 \text{ TB/an}$
- Stratégie: Déplacer données > 90 jours vers Blob Cool Tier

### Année 1:

- Volume moyen: 91 TB (rampe progressive)
- Prix Cool Tier:  $\sim \$0.0184/\text{GB/mois}$
- Coût:  $91,000 \text{ GB} \times \$0.0184 = \mathbf{\$1,674/mois}$

### Année 2 (steady state):

- Volume: 182.5 TB
- Coût:  $182,500 \text{ GB} \times \$0.0184 = \mathbf{\$3,358/mois}$

**Coût moyen Année 1-2: \$2,516/mois**

## Transactions et Operations

### Continuous Export (ADX → Blob):

- Write operations:  $\sim 100,000/\text{jour}$
- Coût write:  $\$0.10/10,000 \text{ ops}$
- Coût mensuel:  $3M \text{ ops} \times \$0.01 = \mathbf{\$30/mois}$

### Read operations (requêtes cold data):

- Estimé: 10,000/jour
- Coût: Marginal ( $\sim \$5/\text{mois}$ )

### Batch Ingestion (si utilisé au lieu Event Hub)

#### Alternative moins chère pour ingestion:

- Stockage temporaire:  $\sim 500 \text{ GB}$  (1 jour buffer)
- Coût:  $500 \text{ GB} \times \$0.0184 = \$9.20/\text{mois}$
- Lifecycle policy: Suppression automatique après 2 jours

**Total Stockage: \$1,674/mois (Année 1) → \$3,358/mois (Année 2)**

---

## 4. Réseau (Network Egress)

### Transferts de Données

#### Intra-Azure:

- Event Hub → ADX (même région): **Gratuit**

- Blob → ADX (même région): **Gratuit**
- ADX → Follower DB (autre région): \$0.05/GB

### Internet Egress (API queries, dashboards):

- Estimé: 2 TB/mois (dashboards, APIs, exports)
- Prix zone 1 (Europe): \$0.087/GB (premiers 10 TB)
- Coût:  $2,000 \text{ GB} \times \$0.087 = \$174/\text{mois}$

### Power BI/Grafana Queries:

- Queries retournent généralement agrégats (petits volumes)
- Estimé: 500 GB/mois
- Coût:  $500 \text{ GB} \times \$0.087 = \$43.50/\text{mois}$

**Total Network: \$500/mois** (estimation conservative)

---

## 5. Monitoring et Management

### Azure Monitor

#### Métriques et Logs:

- Métriques ADX:  $100 \text{ métriques} \times \$0.10 = \$10/\text{mois}$
- Log ingestion (monitoring ADX):  $50 \text{ GB}/\text{mois} \times \$2.76/\text{GB} = \$138/\text{mois}$
- Log retention: 30 jours (standard)

**Total Azure Monitor: \$148/mois**

### Alertes et Actions

#### Alertes:

- ~20 règles d'alerte
- Coût:  $\$0.10/\text{règle}/\text{mois} = \$2/\text{mois}$
- Evaluations: Incluses (premières 1000 gratuites)

#### Action Groups:

- Email notifications: Gratuit
- Logic Apps triggers:  $\sim \$50/\text{mois}$

**Total Alertes: \$52/mois**

### Management Overhead

#### Azure Advisor, Security Center, etc.:

- Standard tier:  $\sim \$200/\text{mois}$
- Inclut: Recommendations, security scanning, compliance

**Total Monitoring & Management: \$400/mois**

---

## 6. Services Additionnels

### **Power BI (Optionnel)**

#### **Power BI Premium Per User:**

- ~10 utilisateurs analystes
- \$20/user/mois = **\$200/mois**

#### **Power BI Embedded** (pour dashboards publics):

- A1 capacity: \$1/heure = ~\$730/mois
- Recommandé si: Dashboards pour >100 users

### **Grafana (Open Source)**

#### **Self-hosted sur AKS:**

- 2 nœuds B4ms: ~\$160/mois
- Stockage: ~\$40/mois
- Total: **\$200/mois**

#### **Grafana Cloud** (Alternative):

- Pro plan: \$299/mois

### **Azure Key Vault**

#### **Stockage de secrets:**

- ~50 secrets
- Coût: \$0.03/10,000 ops
- Total: ~\$5/mois (négligeable)

### **Private Endpoints**

#### **Connexions privées:**

- ADX: \$10/mois
- Event Hub: \$10/mois
- Storage: \$10/mois
- Total: **\$30/mois**

---

## Scénarios de Coûts

Scénario 1: Production Optimisée (Recommandé)

<b>Composant</b>	<b>Coût Mensuel</b>
ADX Cluster (8 × E16s_v5, RI 3 ans)	\$3,835
ADX Hot Storage (90 jours)	\$3,600
Event Hubs (Standard, 10 TU)	\$3,456
Blob Cold Storage	\$1,674
Network	\$500
Monitoring	\$400
<b>TOTAL</b>	<b>\$13,465/mois</b>
<b>ANNUEL</b>	<b>\$161,580</b>

**Avec Power BI et Grafana: \$13,865/mois (\$166,380/an)**

---

## Scénario 2: Startup / Coût Minimum

### Optimisations:

- ADX: 6 nœuds E8s\_v5 (au lieu de 8 × E16s\_v5)
- Event Hub remplacé par Blob batch ingestion
- Hot storage: 30 jours (au lieu de 90)
- Pas de Reserved Instances (flexibilité)

<b>Composant</b>	<b>Coût Mensuel</b>
ADX Cluster (6 × E8s_v5)	\$3,150
ADX Hot Storage (30 jours)	\$1,200
Blob Ingestion + Storage	\$150
Network	\$300
Monitoring	\$200
<b>TOTAL</b>	<b>\$5,000/mois</b>
<b>ANNUEL</b>	<b>\$60,000</b>

### Trade-offs:

- ⚡ Latence ingestion: 5-15 minutes (vs <2 min)
  - ⚡ Moins de ressources pour queries complexes
  - ⚡ Rétention hot réduite (30j vs 90j)
- 

## Scénario 3: Enterprise avec HA Multi-région

### Ajouts:

- Cluster secondaire (follower): 50% du coût primaire
- Geo-replication pour stockage
- Premium Event Hubs
- Multi-region networking

<b>Composant</b>	<b>Coût Mensuel</b>
ADX Cluster Primaire (RI 3 ans)	\$7,435
ADX Cluster Secondaire (follower)	\$3,718
Event Hubs Premium	\$672
Blob Storage (GRS)	\$3,348
Network (multi-region)	\$1,500
Monitoring	\$600
<b>TOTAL</b>	<b>\$17,273/mois</b>
<b>ANNUEL</b>	<b>\$207,276</b>

### Avantages:

- RPO < 15 minutes
- RTO < 1 heure
- Lecture distribuée (performance)
- SLA 99.99%

## Évolution des Coûts sur 3 Ans

Projection (Scénario Production Optimisée)

### Hypothèses:

- Croissance volume: 10% par an
- Pas d'augmentation des prix Azure (conservatif)
- Reserved Instances renouvelées

<b>Année</b>	<b>Volume/Jour</b>	<b>ADX Compute</b>	<b>Storage Total</b>	<b>Event Hub</b>	<b>Total Mensuel</b>	<b>Total Annuel</b>
<b>An 1</b>	5 TB	\$7,435	\$3,600	\$3,456	\$15,291	\$183,492
<b>An 2</b>	5.5 TB	\$7,435	\$7,258	\$3,802	\$18,795	\$225,540
<b>An 3</b>	6 TB	\$8,179	\$10,910	\$4,182	\$23,571	\$282,852

**Note:** An 1 Storage = Hot uniquement; An 2+ inclut accumulation de cold storage.

## Optimisations des Coûts

## 1. Reserved Instances (RI)

### **Impact:**

- 1 an: -38% compute → Économie \$46,000/an
- 3 ans: -62% compute → Économie \$75,000/an

**Recommandation:** RI 3 ans après 6 mois de validation.

## 2. Tiered Storage Strategy

### **Stratégie:**

```
Hot Cache (SSD): 7 jours → Requêtes critiques  
Hot Storage: 30 jours → Requêtes fréquentes  
Cold Storage (Blob): 31-365 jours → Historique  
Archive: >1 an → Compliance
```

**Impact:** Réduction stockage de **40-60%**

## 3. Compression et Formats

### **Parquet vs JSON:**

- Parquet: 50-70% plus petit
- Impact: Réduction Event Hub throughput (moins de TUs)
- Économie estimée: **\$500-800/mois**

## 4. Batch au lieu de Streaming (si acceptable)

### **Remplacement Event Hub par Blob:**

- Économie: ~\$3,456/mois
- Trade-off: Latence 5-15 min (vs <2 min)
- **Recommandé pour:** Dev/test, logs non-critiques

## 5. Query Optimization

### **Materialized Views:**

- Pré-agrégations pour dashboards fréquents
- Réduction compute jusqu'à **70%** pour ces queries

### **Cache Policy Tuning:**

- Identifier queries fréquentes
- Ajuster hot cache (7-14 jours)
- Impact: P95 latency -50%

## 6. Cleanup & Retention Policies

## Aggressive cleanup:

- Logs debug/trace: 30 jours seulement
- Logs info: 90 jours
- Logs error/critical: 1 an

**Impact:** Réduction storage de **30-50%**

## 7. Auto-scaling

### Metrics-based scaling:

- Scale down durant heures creuses
- Économie: **15-25%** (si usage variable)
- Exemple: 8 nœuds peak, 5 nœuds off-hours

## Comparaison avec Alternatives

Azure Data Explorer vs Alternatives

Solution	Coût Mensuel Estimé	Avantages	Inconvénients
<b>ADX</b>	<b>\$13,465-20,340</b>	Performance, KQL, intégration Azure	Coût compute élevé
<b>Elasticsearch (self-hosted AKS)</b>	\$8,000-12,000	Mature, flexible	Gestion complexe, HA difficile
<b>Elastic Cloud</b>	\$15,000-25,000	Managé, ecosystem riche	Coût élevé pour volume
<b>Splunk Cloud</b>	\$30,000-50,000	Très mature, features riches	Très cher (GB ingested)
<b>Datadog Logs</b>	\$25,000-40,000	SaaS, simple	Très cher à scale
<b>Azure Log Analytics</b>	\$18,000-28,000	Intégré Azure Monitor	Moins performant queries complexes
<b>AWS OpenSearch</b>	\$10,000-16,000	Managé, scaling	Pas Azure natif

**Conclusion:** ADX offre le **meilleur ratio performance/coût** pour 5TB/jour avec Azure.

## ROI et Justification

### Coûts Évités

#### Sans solution centralisée:

- Dev time debugging:  $100\text{h}/\text{mois} \times \$100/\text{h} = \$10,000/\text{mois}$
- Downtime costs:  $2\text{h}/\text{mois} \times \$50,000/\text{h} = \$100,000/\text{mois}$  (si e-commerce)
- Incidents non détectés: Incalculable

**Avec ADX:**

- Time to insight: Minutes (vs heures/jours)
- Proactive detection: Éviter incidents
- Capacity planning: Optimiser infra

**ROI estimé:** 3-6 mois pour entreprises moyennes/grandes

## Bénéfices Business

**Quantifiables:**

- ↓ 60% temps debugging (équipes ops/dev)
- ↓ 40% MTTR (Mean Time To Restore)
- ↓ 30% incidents production (détection proactive)

**Non-quantifiables:**

- Meilleure compliance (audit trails)
- Insights business (analytics sur logs applicatifs)
- Confiance clients (moins de downtime)

---

## Recommandations Finales

### Phase 1: Pilot (Mois 1-3)

**Budget:** \$5,000-7,000/mois

- Cluster minimal (6 nœuds E8s\_v5)
- Blob ingestion batch
- 30 jours hot storage
- Pas de RI (flexibilité)

### Phase 2: Production (Mois 4-6)

**Budget:** \$15,000-18,000/mois

- Cluster production (8 nœuds E16s\_v5)
- Event Hubs streaming
- 90 jours hot storage
- Monitoring complet

### Phase 3: Optimisation (Mois 7-12)

**Budget:** \$13,000-15,000/mois

- Reserved Instances 3 ans
- Tiered storage optimisé
- Materialized views
- Auto-scaling

## Phase 4: Scale & HA (Année 2+)

**Budget:** \$15,000-20,000/mois

- Multi-région si besoin
  - Query optimizations
  - ML pour anomaly detection
  - Self-service analytics
- 

## Checklist Budget

### Mensuel

- Azure Data Explorer compute
- ADX hot storage
- Event Hubs ou Blob Storage
- Network egress
- Azure Monitor
- Alertes et actions

### Optionnel

- Power BI licenses
- Grafana hosting
- Private endpoints
- Follower database (DR)
- Support Azure (Standard/Professional)

### Annuel

- Reserved Instances (renouvellement)
  - Storage lifecycle optimization review
  - Capacity planning update
  - Cost optimization audit
- 

## Contacts et Ressources

**Azure Pricing Calculator:** <https://azure.microsoft.com/pricing/calculator/>

**ADX Pricing Details:** <https://azure.microsoft.com/pricing/details/data-explorer/>

**Event Hubs Pricing:** <https://azure.microsoft.com/pricing/details/event-hubs/>

**Storage Pricing:** <https://azure.microsoft.com/pricing/details/storage/blobs/>

**Azure Cost Management:** [https://portal.azure.com/#blade/Microsoft\\_Azure\\_CostManagement](https://portal.azure.com/#blade/Microsoft_Azure_CostManagement)

---

## Conclusion

Pour une solution d'analyse de **5 TB de logs par jour** avec Azure Data Explorer:

**Coût Production Baseline: \$15,000-20,000/mois** (\$180,000-240,000/an)

**Coût Production Optimisé: \$13,000-15,000/mois** (\$156,000-180,000/an)

**Recommandation:** Commencer avec configuration standard, valider pendant 3-6 mois, puis optimiser avec Reserved Instances et tiered storage pour réduire les coûts de **30-40%**.

Le **ROI est positif** dès 3-6 mois pour la majorité des entreprises grâce à la réduction du temps de debugging et la détection proactive d'incidents.

## Stratégie d'Ingestion de Données

---

### Vue d'ensemble

L'ingestion de 5TB de logs par jour (~208 GB/heure en moyenne) nécessite une architecture robuste, scalable et résiliente. Ce document détaille les différentes stratégies d'ingestion pour Azure Data Explorer.

### Volume et Contraintes

#### Métriques Cibles

- **Volume quotidien:** 5 TB (non compressé)
- **Débit moyen:** ~208 GB/heure (~58 MB/seconde)
- **Débit peak:** ~625 GB/heure (facteur 3x aux heures de pointe)
- **Latence souhaitée:** < 2 minutes end-to-end
- **Format principal:** JSON structuré
- **Compression ratio:** 10:1 (5TB → 500GB stocké)

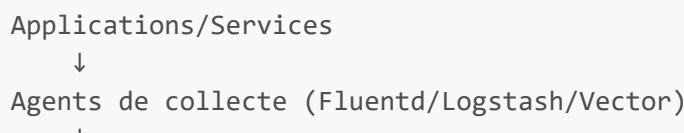
#### Contraintes Techniques

- **Limites ADX:**
  - Ingestion: ~200 GB/heure/nœud
  - Fichiers optimaux: 100MB - 1GB
  - Batch size: 1GB ou 1000 blobs
- **Network:** Bandwidth suffisant (10+ Gbps)
- **Event Hub:** 32+ partitions recommandées

### Options d'Ingestion

Option 1: Azure Event Hubs (Streaming) - ★ RECOMMANDÉ

#### Architecture



```

Azure Event Hubs (32+ partitions)
↓
ADX Data Connection (Native integration)
↓
Azure Data Explorer Tables

```

## Avantages

- **Faible latence:** < 30 secondes à 2 minutes
- **Découplage:** Buffer entre producers et consumers
- **Résilience:** Retry automatique, dead-letter queue
- **Scaling:** Auto-inflate jusqu'à 40 TUs
- **Simplicité:** Gestion automatique par ADX

## Configuration Event Hub

### Namespace:

```

az eventhubs namespace create \
--name "loganalysis-prod-eh" \
--resource-group "rg-loganalysis-prod" \
--location "westeurope" \
--sku "Standard" \
--enable-auto-inflate true \
--maximum-throughput-units 20 \
--capacity 10

```

### Event Hub:

```

az eventhubs eventhub create \
--name "application-logs" \
--namespace-name "loganalysis-prod-eh" \
--partition-count 32 \
--message-retention 1 \
--cleanup-policy "Delete"

```

### Paramètres clés:

- **Partitions:** 32 (permet 32 MB/s = ~115 GB/h)
- **Throughput Units:** 10-20 (1 TU = 1 MB/s ingress)
- **Retention:** 1 jour (suffisant pour buffer)
- **Auto-inflate:** Activé pour peaks

## Calcul du dimensionnement

Débit peak: 625 GB/h = 173 MB/s  
Avec compression 2:1 (avant Event Hub): ~86 MB/s  
Throughput Units requis: 86 TU  
Avec auto-inflate: Commencer à 10 TU, scaler à 20 TU

Partitions pour parallélisme ADX:  
32 partitions × 1 MB/s = 32 MB/s base  
Avec burst: ~100-150 MB/s  
Couverture: 86 MB/s ✓

## Data Connection ADX

```
.create-or-alter table ApplicationLogs ingestion eventhub data connection
'ApplicationLogsConnection'
```json
{
  "properties": {
    "eventHubResourceId": "/subscriptions/{sub-id}/resourceGroups/rg-loganalysis-
prod/providers/Microsoft.EventHub/namespaces/loganalysis-prod-
eh/eventhubs/application-logs",
    "consumerGroup": "adx-consumer",
    "tableName": "ApplicationLogs",
    "mappingRuleName": "ApplicationLogsMapping",
    "dataFormat": "JSON",
    "compression": "GZip",
    "eventSystemProperties": ["x-opt-enqueued-time"],
    "managedIdentityResourceId": "/subscriptions/{sub-id}/resourceGroups/rg-
loganalysis-prod/providers/Microsoft.ManagedIdentity/userAssignedIdentities/adx-
identity"
  }
}
```

## #### Configuration des Agents de Collecte

```
**Fluentd**:
```yaml
<source>
  @type tail
  path /var/log/application/*.log
  pos_file /var/log/td-agent/application.log.pos
  tag application.logs
  format json
  time_key timestamp
  time_format %Y-%m-%dT%H:%M:%S.%LZ
</source>

<filter application.logs>
```

```

@type record_transformer
enable_ruby true
<record>
  hostname "#{Socket.gethostname}"
  environment "#{ENV['ENVIRONMENT']}"
  region "#{ENV['AZURE_REGION']}"
</record>
</filter>

<match application.logs>
  @type azure_event_hubs
  connection_string "#{ENV['EVENT_HUB_CONNECTION_STRING']}"
  hub_name "application-logs"

  # Batching pour efficacité
  <buffer>
    @type file
    path /var/log/td-agent/buffer/eventhub
    flush_mode interval
    flush_interval 10s
    chunk_limit_size 5MB
    total_limit_size 1GB
    retry_type exponential_backoff
    retry_max_interval 30s
  </buffer>

  # Compression
  compress gzip
</match>

```

### Vector (Alternative moderne):

```

[sources.application_logs]
type = "file"
include = ["/var/log/application/*.log"]
data_dir = "/var/lib/vector"

[transforms.parse_json]
type = "remap"
inputs = ["application_logs"]
source = ''
  . = parse_json!(.message)
  .hostname = get_hostname_()
  .environment = get_env_var!("ENVIRONMENT")
  ...

[sinks.azure_event_hub]
type = "azure_event_hubs"
inputs = ["parse_json"]
connection_string = "${EVENT_HUB_CONNECTION_STRING}"
hub_name = "application-logs"
encoding.codec = "json"

```

```
# Batching
batch.max_bytes = 5242880 # 5MB
batch.timeout_secs = 10

# Compression
compression = "gzip"
```

## Monitoring Event Hub

```
// Métriques Event Hub dans ADX
.show commands
| where CommandType == "DataIngestPull"
| where OriginalClientRequestId contains "eventhub"
| summarize
    EventCount = sum(RowCount),
    TotalSizeGB = sum(ResourceUtilization.MemoryPeak) / 1024 / 1024 / 1024,
    AvgLatencySeconds = avg(Duration) / 1000.0
    by bin(StartedOn, 5m)
| render timechart
```

## Option 2: Azure Blob Storage (Batch) - Pour Historique

### Architecture

```
Applications/Services
  ↓
Agents de collecte
  ↓
Azure Blob Storage (hourly batches)
  ↓
Event Grid Notification
  ↓
ADX Data Connection
  ↓
Azure Data Explorer Tables
```

### Avantages

- **Coût réduit:** Pas de Event Hub
- **Simplicité:** Drop de fichiers
- **Réingestion:** Facile à rejouer
- **Formats variés:** JSON, Parquet, CSV, Avro

### Inconvénients

- ⚠️ **Latence plus élevée:** 5-15 minutes
- ⚠️ **Moins de résilience:** Pas de buffer

## Configuration Storage Account

```
az storage account create \
--name "loganalyticsprods" \
--resource-group "rg-loganalysis-prod" \
--location "westeurope" \
--sku "Standard_LRS" \
--kind "StorageV2" \
--access-tier "Hot" \
--enable-hierarchical-namespace true # Data Lake Gen2

az storage container create \
--account-name "loganalyticsprods" \
--name "application-logs" \
--public-access off
```

## Event Grid Configuration

```
# Event Grid Topic
az eventgrid topic create \
--name "storage-events-topic" \
--resource-group "rg-loganalysis-prod" \
--location "westeurope"

# Event Grid Subscription
az eventgrid event-subscription create \
--name "blob-created-subscription" \
--source-resource-id "/subscriptions/{sub-id}/resourceGroups/rg-loganalysis-
prod/providers/Microsoft.Storage/storageAccounts/loganalyticsprods" \
--endpoint-type "azurefunction" \
--endpoint "/subscriptions/{sub-id}/resourceGroups/rg-loganalysis-
prod/providers/Microsoft.Web/sites/adx-ingestion-
func/functions/BlobCreatedHandler" \
--included-event-types "Microsoft.Storage.BlobCreated"
```

## Data Connection ADX

```
.create-or-alter table ApplicationLogs ingestion blob data connection
'StorageConnection'
```json
{
  "properties": {
    "storageAccountResourceId": "/subscriptions/{sub-id}/resourceGroups/rg-
loganalysis-prod/providers/Microsoft.Storage/storageAccounts/loganalyticsprods",
```

```
"containerName": "application-logs",
"eventGridResourceId": "/subscriptions/{sub-id}/resourceGroups/rg-loganalysis-
prod/providers/Microsoft.EventGrid/topics/storage-events-topic",
"tableName": "ApplicationLogs",
"mappingRuleName": "ApplicationLogsMapping",
"dataFormat": "Parquet",
"ignoreFirstRecord": false,
"managedIdentityResourceId": "/subscriptions/{sub-id}/resourceGroups/rg-
loganalysis-prod/providers/Microsoft.ManagedIdentity/userAssignedIdentities/adx-
identity"
}
}
```

#### #### Pattern de Naming des Fichiers

```
application-logs/ └── year=2024/ └── month=06/ └── day=15/ └── hour=14/ ├── app-logs-20240615-
140000-001.parquet.gz └── app-logs-20240615-140000-002.parquet.gz └── app-logs-20240615-140000-
003.parquet.gz
```

\*\*Best practices\*\*:

- \*\*Taille\*\*: 100-500 MB par fichier (après compression)
- \*\*Format\*\*: Parquet (meilleure compression et performance)
- \*\*Compression\*\*: GZip ou Snappy
- \*\*Fréquence\*\*: Toutes les 5-10 minutes

#### #### Script de Upload

```
```python
from azure.storage.blob import BlobServiceClient
import gzip
import json
from datetime import datetime
import pyarrow as pa
import pyarrow.parquet as pq

class LogUploader:
    def __init__(self, connection_string, container_name):
        self.blob_service =
            BlobServiceClient.from_connection_string(connection_string)
        self.container_client =
            self.blob_service.get_container_client(container_name)

    def upload_logs_parquet(self, logs, partition_date):
        # Conversion en Parquet
        table = pa.Table.from_pylist(logs)
```

```

# Path avec partitioning
year = partition_date.strftime("%Y")
month = partition_date.strftime("%m")
day = partition_date.strftime("%d")
hour = partition_date.strftime("%H")
timestamp = partition_date.strftime("%Y%m%d-%H%M%S")

blob_path = f"year={year}/month={month}/day={day}/hour={hour}/logs-{timestamp}.parquet.gz"

# Write to buffer
import io
buffer = io.BytesIO()
pq.write_table(table, buffer, compression='gzip')
buffer.seek(0)

# Upload
blob_client = self.container_client.get_blob_client(blob_path)
blob_client.upload_blob(buffer, overwrite=False)

print(f"Uploaded {len(logs)} logs to {blob_path}")

# Usage
uploader = LogUploader(connection_string, "application-logs")
uploader.upload_logs_parquet(log_batch, datetime.now())

```

## Option 3: Hybrid (Event Hub + Storage)

### Use Cases

- **Event Hub:** Logs critiques temps réel (erreurs, sécurité)
- **Storage:** Logs bulk non-critiques (debug, trace)

### Routage par niveau de严重性

```

def route_log(log_entry):
    level = log_entry.get('level', 'info').lower()

    if level in ['error', 'critical', 'warning']:
        # Route vers Event Hub pour traitement immédiat
        send_to_event_hub(log_entry)
    else:
        # Route vers Storage pour batch processing
        send_to_storage(log_entry)

```

### Configuration Fluentd avec Routing

```
<match application.logs>
  @type copy

  # Logs critiques vers Event Hub
  <store>
    @type azure_event_hubs
    connection_string "${ENV['EVENT_HUB_CONNECTION_STRING']}"
    hub_name "critical-logs"
    <filter>
      @type grep
      <regexp>
        key level
        pattern /(error|critical|warning)/i
      </regexp>
    </filter>
  </store>

  # Tous les logs vers Storage (batch)
  <store>
    @type azure_storage_append_blob
    azure_storage_account "${ENV['STORAGE_ACCOUNT']}"
    azure_storage_access_key "${ENV['STORAGE_KEY']}"
    azure_container "application-logs"
    path "logs/%Y/%m/%d/%H/logs-#${Socket.gethostname()}-%M.json"

    <buffer time>
      @type file
      path /var/log/td-agent/buffer/storage
      timekey 600 # 10 minutes
      timekey_wait 60
      chunk_limit_size 256MB
    </buffer>
  </store>
</match>
```

---

## Formats de Données

JSON (Recommandé pour démarrage)

**Avantages:**

- Flexible et facile à débugger
- Support natif de dynamic types dans ADX
- Lisible par humains

**Inconvénients:**

- Taille plus grande (~30-40% vs Parquet)
- Parsing moins performant

**Exemple:**

```
{
  "timestamp": "2024-06-15T14:35:42.123Z",
  "level": "error",
  "logger": "com.company.OrderService",
  "message": "Failed to process order",
  "exception": "OrderProcessingException: Timeout while contacting payment service",
  "traceId": "abc123-def456-ghi789",
  "spanId": "span-001",
  "serviceName": "OrderService",
  "environment": "production",
  "properties": {
    "orderId": "ORD-123456",
    "userId": "user-789",
    "amount": 99.99,
    "retryCount": 3
  }
}
```

**Parquet (Recommandé pour production)****Avantages:**

- Compression excellente (50-70% vs JSON)
- Lecture columnaire très rapide
- Schéma fort et typé

**Inconvénients:**

- Nécessite conversion côté agent
- Moins flexible pour schémas changeants

**Configuration:**

```
import pyarrow as pa
import pyarrow.parquet as pq

# Définition du schéma
schema = pa.schema([
    ('timestamp', pa.timestamp('ms')),
    ('level', pa.string()),
    ('logger', pa.string()),
    ('message', pa.string()),
    ('exception', pa.string()),
    ('traceId', pa.string()),
    ('serviceName', pa.string()),
    ('properties', pa.string()) # JSON stringifié
])
```

```
# Écriture
pq.write_table(
    table,
    'logs.parquet',
    compression='snappy',
    use_dictionary=True
)
```

CSV (Pour simplicité)

**Use case:** Logs simples, outils legacy

**Format:**

```
timestamp,level,serviceName,message
2024-06-15T14:35:42Z,error,OrderService,"Failed to process order"
```

## Transformation et Enrichissement

Update Policy pour Enrichissement

```
// Fonction d'enrichissement
.create-or-alter function EnrichLogs() {
    ApplicationLogs
    | extend
        // Parsing enrichi
        Severity = case(
            Level == "critical", 5,
            Level == "error", 4,
            Level == "warning", 3,
            Level == "info", 2,
            1
        ),
        // Extraction de métadonnées
        ErrorCode = extract(@"errorCode=(\d+)", 1, Message),
        IPAddress = extract(@"ip=([0-9.]+)", 1, Message),
        // Géolocalisation
        GeoInfo = geo_info_from_ip_address(extract(@"ip=([0-9.]+)", 1, Message)),
        // Catégorisation
        ServiceCategory = extract(@"^(^-]+)", 1, ServiceName),
        // Date parts pour partitioning
        Year = getyear(Timestamp),
        Month = getmonth(Timestamp),
        Day = dayofmonth(Timestamp),
        Hour = hourofday(Timestamp)
    | extend
        Country = tostring(GeoInfo.country),
```

```

    City = tostring(GeoInfo.city)
    | project-away GeoInfo
}

// Table enrichie
.create table EnrichedApplicationLogs (
    Timestamp: datetime,
    Level: string,
    Severity: int,
    ServiceName: string,
    ServiceCategory: string,
    Message: string,
    ErrorCode: string,
    IPAddress: string,
    Country: string,
    City: string,
    TraceId: string,
    Year: int,
    Month: int,
    Day: int,
    Hour: int
)

// Update policy
.alter table EnrichedApplicationLogs policy update
@'[{{
    "Source": "ApplicationLogs",
    "Query": "EnrichLogs()", 
    "IsEnabled": true,
    "IsTransactional": true
}]'

```

## Monitoring et Troubleshooting

### Dashboard d'Ingestion

```

// Vue d'ensemble de l'ingestion
.show commands
| where CommandType in ("DataIngestPull", "DataIngestPush")
| where StartedOn > ago(1h)
| summarize
    IngestedRows = sum(RowCount),
    IngestedGB = sum(ResourceUtilization.MemoryPeak) / 1024 / 1024 / 1024,
    AvgLatencySeconds = avg(Duration) / 1000.0,
    P95LatencySeconds = percentile(Duration, 95) / 1000.0,
    FailedCount = countif(State == "Failed")
    by bin(StartedOn, 5m)
| render timechart

```

## Détection d'Échecs

```
// Échecs d'ingestion récents
.show ingestion failures
| where FailedOn > ago(1h)
| project
    FailedOn,
    Database,
    Table,
    OperationId,
    ErrorCode,
    FailureStatus,
    Details
| order by FailedOn desc
```

## Alertes Recommandées

### Azure Monitor Alerts:

#### 1. Ingestion Rate Drop

```
// Alert si ingestion < 50% de la moyenne
let baseline = ApplicationLogs
| where ingestion_time() > ago(7d)
| summarize AvgRowsPerMinute = count() / (7 * 24 * 60);
ApplicationLogs
| where ingestion_time() > ago(5m)
| summarize CurrentRowsPerMinute = count() / 5
| extend Baseline = toscalar(baseline)
| where CurrentRowsPerMinute < (Baseline * 0.5)
```

#### 2. Ingestion Latency High

```
// Alert si latence > 5 minutes
ApplicationLogs
| where ingestion_time() > ago(10m)
| extend IngestionLatency = ingestion_time() - Timestamp
| summarize P95Latency = percentile(IngestionLatency, 95)
| where P95Latency > 5m
```

#### 3. Failed Ingestions

```
// Alert si échecs > 1% du volume
.show ingestion failures
| where FailedOn > ago(5m)
| summarize FailureCount = count()
```

```
| extend Threshold = 100 // Ajuster selon volume  
| where FailureCount > Threshold
```

---

## Best Practices

### Sizing et Performance

1. **Taille des fichiers:** 100MB - 1GB (optimal)
2. **Batch frequency:** 5-10 minutes
3. **Compression:** Toujours activer (GZip ou Snappy)
4. **Partitions Event Hub:** Au moins 1 partition par 1 MB/s

### Résilience

1. **Retry Logic:** Exponentiel backoff
2. **Dead Letter Queue:** Pour messages non ingérables
3. **Monitoring continu:** Alertes sur échecs/latence
4. **Backfill capability:** Capacité de réingérer depuis Storage

### Sécurité

1. **Managed Identities:** Pas de secrets dans config
2. **Private Endpoints:** Connexions privées
3. **Encryption:** At rest et in transit
4. **Audit Logging:** Tracer toutes opérations d'ingestion

### Coûts

1. **Compression:** Réduire network et storage costs
2. **Batch vs Streaming:** Storage moins cher que Event Hub
3. **Reserved Capacity:** Pour Event Hub et ADX
4. **Lifecycle policies:** Archiver logs anciens

---

## Checklist de Déploiement

### Phase 1: Setup Infrastructure

- Créer cluster ADX
- Créer Event Hub / Storage Account
- Configurer Event Grid (si Storage)
- Setup Managed Identities
- Configurer Private Endpoints

### Phase 2: Configuration ADX

- Créer Database et Tables
- Définir Ingestion Mappings

- Configurer Data Connections
- Setup Update Policies
- Configurer Retention Policies

### Phase 3: Agents et Collection

- Déployer agents de collecte
- Configurer routing et buffering
- Tester ingestion end-to-end
- Valider formats et mappings

### Phase 4: Monitoring

- Configurer dashboards d'ingestion
- Setup alertes critiques
- Documenter runbooks
- Tester procédures d'incident

### Phase 5: Optimization

- Tuner batch sizes
- Optimiser compression
- Ajuster capacités
- Réviser coûts

---

## Ressources

- [ADX Data Ingestion Overview](#)
- [Event Hub Best Practices](#)
- [Fluentd Documentation](#)
- [Vector Documentation](#)

---

## Exemples KQL pour Analyse de Logs

### Introduction à KQL (Kusto Query Language)

KQL est un langage de requête optimisé pour l'analyse de grands volumes de données, particulièrement efficace pour les logs et télémétries. Ce document présente des exemples pratiques pour l'analyse quotidienne de logs.

### Structure de Base d'une Requête KQL

```
TableName  
| where Condition  
| extend NewColumn = Expression  
| summarize Aggregation by GroupBy
```

```
| order by Column  
| take N
```

## Exemples par Cas d'Usage

### 1. Analyses de Base

#### **Compter les logs par niveau de严重性**

```
ApplicationLogs  
| where Timestamp > ago(24h)  
| summarize Count = count() by Level  
| order by Count desc  
| render piechart
```

#### **Top 10 services les plus verbeux**

```
ApplicationLogs  
| where Timestamp > ago(1h)  
| summarize LogCount = count() by ServiceName  
| top 10 by LogCount desc
```

#### **Logs d'erreur avec contexte**

```
ApplicationLogs  
| where Level in ("Error", "Critical")  
| where Timestamp > ago(1h)  
| project Timestamp, ServiceName, Message, Exception, TraceId  
| order by Timestamp desc  
| take 100
```

### 2. Analyse Temporelle

#### **Distribution horaire des logs**

```
ApplicationLogs  
| where Timestamp > ago(7d)  
| summarize Count = count() by bin(Timestamp, 1h), Level  
| render timechart
```

#### **Comparaison jour vs jour**

```
ApplicationLogs
| where Timestamp > ago(48h)
| extend DayCategory = iff(Timestamp > ago(24h), "Today", "Yesterday")
| summarize Count = count() by bin(Timestamp, 1h), DayCategory
| render timechart
```

## Pattern hebdomadaire

```
ApplicationLogs
| where Timestamp > ago(30d)
| extend DayOfWeek = dayofweek(Timestamp)
| extend Hour = hourofday(Timestamp)
| summarize AvgLogs = avg(todouble(1)) by DayOfWeek, Hour
| render heatmap
```

## 3. Performance et Latence

### Analyse de latence par endpoint

```
ApplicationLogs
| where ServiceName == "ApiGateway"
| where Timestamp > ago(1h)
| extend Endpoint = extract(@"endpoint=([^ ]+)", 1, Message)
| summarize
    RequestCount = count(),
    P50 = percentile(DurationMs, 50),
    P95 = percentile(DurationMs, 95),
    P99 = percentile(DurationMs, 99),
    Max = max(DurationMs)
    by Endpoint
| where RequestCount > 100
| order by P95 desc
```

### Requêtes lentes (SLO > 3 secondes)

```
ApplicationLogs
| where DurationMs > 3000
| where Timestamp > ago(24h)
| extend Endpoint = extract(@"path=([^ ]+)", 1, RequestPath)
| summarize
    SlowRequests = count(),
    AvgDuration = avg(DurationMs),
    MaxDuration = max(DurationMs)
```

```
by Endpoint, ServiceName
| order by SlowRequests desc
```

## Time series de latence avec détection d'anomalies

```
ApplicationLogs
| where ServiceName == "PaymentService"
| where Timestamp > ago(7d)
| summarize AvgDuration = avg(DurationMs) by bin(Timestamp, 5m)
| extend (anomalies, score, baseline) = series_decompose_anomalies(AvgDuration,
1.5)
| render anomalychart with (anomalycolumns=anomalies)
```

## 4. Analyse d'Erreurs

### Top erreurs par fréquence

```
ApplicationLogs
| where Level == "Error"
| where Timestamp > ago(24h)
| extend ErrorType = extract(@"Exception: ([^:]+)", 1, Exception)
| summarize
    ErrorCode = count(),
    AffectedUsers = dcount(UserId),
    AffectedServices = dcount(ServiceName),
    FirstOccurrence = min(Timestamp),
    LastOccurrence = max(Timestamp),
    SampleMessage = any(Message)
    by ErrorType
| order by ErrorCode desc
| take 20
```

### Cascade d'erreurs (erreurs liées par TraceId)

```
let ErrorTraces = ApplicationLogs
| where Level == "Error" and Timestamp > ago(1h)
| where ServiceName == "OrderService"
| distinct TraceId;
ApplicationLogs
| where TraceId in (ErrorTraces)
| order by TraceId, Timestamp asc
| project Timestamp, TraceId, ServiceName, Level, Message, DurationMs
```

### Taux d'erreur par service

```

ApplicationLogs
| where Timestamp > ago(24h)
| summarize
    TotalRequests = count(),
    ErrorCount = countif(Level in ("Error", "Critical")),
    ErrorRate = round(countif(Level in ("Error", "Critical")) * 100.0 / count(), 2)
2)
    by ServiceName, bin(Timestamp, 1h)
| where TotalRequests > 100
| order by ErrorRate desc
| render timechart

```

## 5. Sécurité et Audit

### Tentatives de connexion échouées

```

ApplicationLogs
| where Logger contains "Auth"
| where Message has_any ("login failed", "authentication failed")
| where Timestamp > ago(1h)
| extend IPAddress = extract(@"ip=([0-9.]+)", 1, Message)
| summarize
    FailedAttempts = count(),
    UniqueUsers = dcount(UserId),
    FirstAttempt = min(Timestamp),
    LastAttempt = max(Timestamp)
    by IPAddress
| where FailedAttempts > 5
| order by FailedAttempts desc

```

### Détection d'attaques par force brute

```

ApplicationLogs
| where Timestamp > ago(5m)
| where ResponseCode in (401, 403)
| extend IPAddress = extract(@"ip=([0-9.]+)", 1, Message)
| summarize
    Attempts = count(),
    TargetedAccounts = make_set(UserId, 10)
    by IPAddress, bin(Timestamp, 1m)
| where Attempts > 10
| project Timestamp, IPAddress, Attempts, TargetedAccounts

```

### Accès à ressources sensibles

```

ApplicationLogs
| where Timestamp > ago(7d)
| where RequestPath has_any ("/admin", "/api/sensitive", "/internal")
| summarize
    AccessCount = count(),
    UniqueUsers = dcount(UserId),
    SuccessfulAccess = countif(ResponseCode < 400),
    BlockedAccess = countif(ResponseCode >= 400)
    by UserId, RequestPath
| where AccessCount > 0
| order by AccessCount desc

```

## 6. Analyse Multi-Service (Distributed Tracing)

### **Vue complète d'une transaction**

```

let traceId = "abc123-def456-ghi789";
ApplicationLogs
| where TraceId == traceId
| order by Timestamp asc
| project
    Timestamp,
    ServiceName,
    Level,
    Message,
    DurationMs,
    ResponseCode,
    Step = row_number()
| extend TotalDuration = toscalar(
    ApplicationLogs
    | where TraceId == traceId
    | summarize max(Timestamp) - min(Timestamp)
    ) / 1ms

```

### **Services les plus lents dans la chaîne**

```

ApplicationLogs
| where Timestamp > ago(1h)
| where isnotempty(TraceId)
| summarize ServiceDuration = sum(DurationMs) by TraceId, ServiceName
| summarize
    AvgDuration = avg(ServiceDuration),
    P95Duration = percentile(ServiceDuration, 95),
    CallCount = count()
    by ServiceName
| order by P95Duration desc

```

## Traçage des dépendances entre services

```

ApplicationLogs
| where Timestamp > ago(1h)
| where isnotempty(TraceId)
| summarize by TraceId, ServiceName
| join kind=inner (
    ApplicationLogs
    | where Timestamp > ago(1h)
    | where isnotempty(TraceId)
    | summarize by TraceId, ServiceName
) on TraceId
| where ServiceName != ServiceName1
| summarize Count = count() by Caller = ServiceName, Callee = ServiceName1
| where Count > 10
| render force_directed_graph

```

## 7. Analyse Business

### Utilisateurs les plus actifs

```

ApplicationLogs
| where Timestamp > ago(24h)
| where isnotempty(UserId)
| summarize
    Actions = count(),
    UniqueEndpoints = dcount(RequestPath),
    TotalDataTransferred = sum(BytesSent + BytesReceived),
    AvgResponseTime = avg(DurationMs),
    ErrorCount = countif(ResponseCode >= 400)
    by UserId
| order by Actions desc
| take 50

```

### Analyse géographique du trafic

```

ApplicationLogs
| where Timestamp > ago(7d)
| extend IPAddress = extract(@"ip=(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})", 1, Message)
| extend GeoInfo = geo_info_from_ip_address(IPAddress)
| extend Country = tostring(GeoInfo.country), City = tostring(GeoInfo.city)
| where isnotempty(Country)
| summarize
    Requests = count(),
    UniqueUsers = dcount(UserId),
    AvgLatency = avg(DurationMs)
    by Country, City

```

```
| order by Requests desc
| take 20
```

## Funnel d'utilisation

```
let StartEvent = "page_view:home";
let Step1 = "page_view:product";
let Step2 = "action:add_to_cart";
let Step3 = "action:checkout";
let users_started = ApplicationLogs
    | where Timestamp > ago(7d)
    | where Message has StartEvent
    | distinct UserId;
let users_step1 = ApplicationLogs
    | where Timestamp > ago(7d)
    | where Message has Step1
    | distinct UserId;
let users_step2 = ApplicationLogs
    | where Timestamp > ago(7d)
    | where Message has Step2
    | distinct UserId;
let users_step3 = ApplicationLogs
    | where Timestamp > ago(7d)
    | where Message has Step3
    | distinct UserId;
print
    Started = toscalar(users_started | count),
    ViewedProduct = toscalar(users_step1 | count),
    AddedToCart = toscalar(users_step2 | count),
    Checkout = toscalar(users_step3 | count)
```

## 8. Monitoring et Alerting

### SLA monitoring (availability)

```
ApplicationLogs
| where Timestamp > ago(30d)
| where ServiceName == "ApiGateway"
| summarize
    TotalRequests = count(),
    SuccessfulRequests = countif(ResponseCode < 500),
    Availability = round(countif(ResponseCode < 500) * 100.0 / count(), 3)
    by bin(Timestamp, 1d)
| extend SLAMet = Availability >= 99.9
| project Timestamp, Availability, SLAMet
```

### Détection de pics anormaux

```

ApplicationLogs
| where Timestamp > ago(30d)
| summarize RequestCount = count() by bin(Timestamp, 5m)
| extend
    Baseline = series_stats_dynamic(RequestCount).avg,
    StdDev = series_stats_dynamic(RequestCount).stdev
| extend Threshold = Baseline + (3 * StdDev)
| where RequestCount > Threshold
| project Timestamp, RequestCount, Baseline, Threshold

```

## Health check des services

```

ApplicationLogs
| where Timestamp > ago(5m)
| summarize LastSeen = max(Timestamp) by ServiceName
| extend
    Status = case(
        LastSeen > ago(2m), "Healthy",
        LastSeen > ago(5m), "Warning",
        "Critical"
    ),
    MinutesSinceLastLog = datetime_diff('minute', now(), LastSeen)
| project ServiceName, Status, MinutesSinceLastLog, LastSeen
| order by Status desc, MinutesSinceLastLog desc

```

## 9. Optimisation des Coûts

### Top services par volume de logs

```

ApplicationLogs
| where Timestamp > ago(30d)
| summarize
    LogCount = count(),
    EstimatedSizeGB = (count() * 500.0) / 1024 / 1024 / 1024 // Assuming ~500
    bytes/log
    by ServiceName
| order by EstimatedSizeGB desc
| extend CumulativePercentage = round(100.0 * row_cumsum(EstimatedSizeGB) /
    toscalar(summarize sum(EstimatedSizeGB)), 2)

```

### Logs de faible valeur (candidats pour rétention réduite)

```

ApplicationLogs
| where Level == "Debug" or Level == "Trace"

```

```

| where Timestamp > ago(7d)
| summarize
  Count = count(),
  SizeEstimateGB = (count() * 500.0) / 1024 / 1024 / 1024
  by ServiceName, Level
| order by SizeEstimateGB desc

```

## 10. Requêtes Avancées

### Analyse de patterns avec regex

```

ApplicationLogs
| where Timestamp > ago(24h)
| extend
  OrderId = extract(@"orderId=([A-Z0-9-]+)", 1, Message),
  Amount = todouble(extract(@"amount=(\d+\.\?\d*)", 1, Message))
| where isnotempty(OrderId)
| summarize
  TotalAmount = sum(Amount),
  OrderCount = dcount(OrderId)
by ServiceName

```

### Join avec table de référence

```

// Table de référence des services
let ServiceMetadata = datatable(ServiceName:string, Team:string, Tier:string)
[
    "AuthService", "Platform", "Critical",
    "PaymentService", "Finance", "Critical",
    "NotificationService", "Communication", "Standard"
];
ApplicationLogs
| where Timestamp > ago(1h)
| summarize ErrorCount = countif(Level == "Error") by ServiceName
| join kind=leftouter ServiceMetadata on ServiceName
| where Tier == "Critical"
| order by ErrorCount desc

```

### Pivot dynamique

```

ApplicationLogs
| where Timestamp > ago(24h)
| summarize Count = count() by Level, ServiceName
| evaluate pivot(Level, sum(Count))

```

## Analyse de séries temporelles avec prédition

```
ApplicationLogs
| where Timestamp > ago(30d)
| where ServiceName == "ApiGateway"
| summarize RequestCount = count() by bin(Timestamp, 1h)
| extend (predictions, pred_score) = series_decompose_forecast(RequestCount, 24)
// Prédire 24h
| render timechart
```

## Fonctions Personnalisées

### Fonction pour standardiser les logs

```
.create-or-alter function ParseApplicationLog(logLevel:string) {
    ApplicationLogs
    | where Level == logLevel
    | extend
        NormalizedTimestamp = bin(Timestamp, 1m),
        ErrorCategory = case(
            Exception contains "Timeout", "Timeout",
            Exception contains "Connection", "Connection",
            Exception contains "Authentication", "Auth",
            "Other"
        )
    | project
        NormalizedTimestamp,
        ServiceName,
        ErrorCategory,
        Message,
        TraceId
}

// Utilisation
ParseApplicationLog("Error")
| where NormalizedTimestamp > ago(1h)
| summarize count() by ErrorCategory
```

### Fonction pour calcul de SLI

```
.create-or-alter function CalculateSLI(serviceName:string, windowSize:timespan) {
    ApplicationLogs
    | where ServiceName == serviceName
    | where Timestamp > ago(windowSize)
    | summarize
        TotalRequests = count(),
        GoodRequests = countif(ResponseCode < 500 and DurationMs < 1000),
```

```

    ErrorRequests = countif(ResponseCode >= 500),
    SlowRequests = countif(DurationMs >= 1000)
| extend
    SLI = round(todouble(GoodRequests) * 100.0 / TotalRequests, 2),
    ErrorRate = round(todouble(ErrorRequests) * 100.0 / TotalRequests, 2),
    SlowRate = round(todouble(SlowRequests) * 100.0 / TotalRequests, 2)
| project SLI, ErrorRate, SlowRate, TotalRequests
}

// Utilisation
CalculateSLI("PaymentService", 1h)

```

## Best Practices KQL

### Performance

1. **Filtrer tôt:** Toujours utiliser `where` avant `summarize` ou `join`
2. **Limiter les colonnes:** Utiliser `project` pour réduire les données
3. **Utiliser le cache:** Requêtes sur hot cache (< 14 jours par défaut)
4. **Éviter les scans complets:** Toujours spécifier une plage temporelle

### Lisibilité

1. **Indentation:** Une opération par ligne
2. **Variables:** Utiliser `let` pour requêtes complexes
3. **Commentaires:** Documenter la logique métier
4. **Naming:** Noms de colonnes explicites

### Exemple de requête bien structurée

```

// Analyse des erreurs critiques avec impact business
// Author: Platform Team | Date: 2024-06-15
let timeRange = 24h;
let errorThreshold = 100;
let criticalServices = dynamic(["PaymentService", "AuthService", "OrderService"]);
//

ApplicationLogs
| where Timestamp > ago(timeRange)                                // Filtrage temporel
| where Level in ("Error", "Critical")                            // Filtrage par严重性
| where ServiceName in (criticalServices)                         // Services critiques
uniquelement
| extend ErrorCategory = extract(@"^(\\w+)Exception", 1, Exception)
| summarize
    ErrorCount = count(),
    AffectedUsers = dcount(UserId),
    UniqueErrors = dcount(ErrorCategory),
    SampleMessages = make_set(Message, 3)
    by ServiceName, ErrorCategory
| where ErrorCount > errorThreshold                               // Seuil significatif
| order by ErrorCount desc

```

```

| project
  ServiceName,
  ErrorCategory,
  ErrorCode,
  AffectedUsers,
  SampleMessages

```

## Ressources Complémentaires

- [KQL Reference officielle](#)
- [KQL Cheat Sheet](#)
- [Best Practices KQL](#)
- [Tutoriels interactifs](#)

## Playground pour Pratique

Microsoft propose un cluster de démonstration avec données samples:

- URL: <https://dataexplorer.azure.com/>
- Cluster: help
- Database: Samples
- Tables: StormEvents, ContosoSales, etc.

Testez vos requêtes avant de les déployer en production!

## Executive Summary - Solution d'Analyse de Logs 5TB/Jour

---

### Objectif

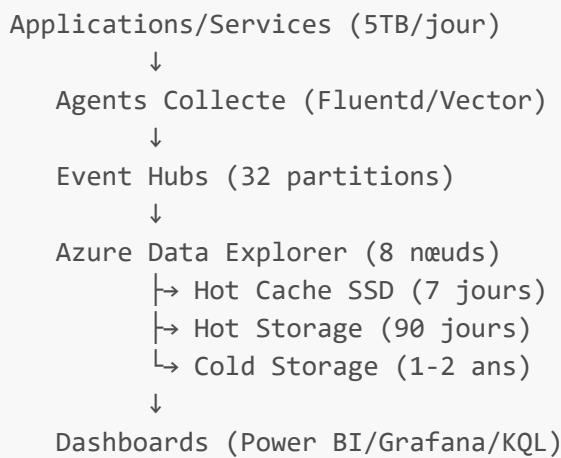
Analyser **5 TB de logs par jour** (~1.8 PB/an) avec des requêtes sub-secondes, latence d'ingestion < 2 minutes, et coûts optimisés.

### Solution Recommandée: Azure Data Explorer (ADX)

Pourquoi ADX?

Critère	ADX	Alternatives
<b>Performance Queries</b>	★★★★★ Sub-seconde	★★★★ Seconde
<b>Latence Ingestion</b>	★★★★★ < 2 min	★★★★ 5-15 min
<b>Compression</b>	★★★★★ 10:1	★★★★ 5:1
<b>Coût</b>	★★★★ Optimal	★★★★ 2-3x plus cher
<b>Scalabilité</b>	★★★★★ Quasi-illimitée	★★★★ Complexe
<b>Facilité</b>	★★★★ KQL simple	★★★★ Varies

## 📊 Architecture en 5 Minutes



## ⌚ Coûts Mensuels (Résumé)

Scénario	Coût/Mois	Coût/An	Use Case
<b>Minimal (Dev/Test)</b>	\$5,000	\$60,000	POC, développement
<b>Production Standard</b>	\$15,291	\$183,492	Production baseline
<b>Optimisé (RI 3 ans)</b>	\$13,465	\$161,580	★ RECOMMANDÉ
<b>Enterprise HA</b>	\$17,273	\$207,276	Multi-région, SLA 99.99%

Détail Coût Production Optimisée (\$13,465/mois)

ADX Compute (RI 3 ans):	\$3,835	(28%)
ADX Hot Storage:	\$3,600	(27%)
Event Hubs:	\$3,456	(26%)
Blob Cold Storage:	\$1,674	(12%)
Network:	\$500	(4%)
Monitoring:	\$400	(3%)
<b>TOTAL:</b>	<b>\$13,465</b>	

## ☒ Spécifications Techniques

Cluster ADX

Composant	Spécification
<b>Nœuds</b>	8 × Standard_E16s_v5
<b>vCPUs</b>	128 cores total
<b>RAM</b>	1 TB total

Composant	Spécification
<b>Hot Cache SSD</b>	4 TB (7-14 jours)
<b>Capacité Ingestion</b>	1,600 GB/h (marge 3x)
<b>Availability</b>	3 Availability Zones

## Stockage

Tier	Rétention	Volume	Latence Query
<b>Hot Cache</b>	7-14 jours	350-700 GB	100-500 ms
<b>Hot Storage</b>	90 jours	3-4.5 TB	500ms-2s
<b>Cold Storage</b>	1-2 ans	18-36 TB	2-5s

## Ingestion

Méthode	Latence	Débit	Coût/Mois
<b>Event Hubs</b> (★ Recommandé)	< 2 min	208 GB/h	\$3,456
<b>Blob Storage</b>	5-15 min	Illimité	\$150
<b>Hybrid</b>	Variable	Mix	\$1,800

## 📈 Performance Garanties

Métrique	Valeur
<b>Query Latency (hot cache)</b>	< 500 ms
<b>Query Latency (hot storage)</b>	< 2 secondes
<b>Ingestion Latency</b>	< 2 minutes
<b>Compression Ratio</b>	10:1 (5TB → 500GB)
<b>Concurrent Queries</b>	100+
<b>SLA Availability</b>	99.9% (99.99% multi-région)

## 🔑 Fonctionnalités Clés

### Ingestion

- ✓ Streaming temps réel (Event Hubs)
- ✓ Batch processing (Blob Storage)
- ✓ Formats multiples (JSON, Parquet, CSV, Avro)
- ✓ Compression automatique (10:1)
- ✓ Transformation à l'ingestion (Update Policies)

## Querying (KQL)

- Langage SQL-like puissant et intuitif
- Time series analysis natives
- Anomaly detection intégrée
- Joins multi-tables
- Agrégations complexes
- Visualisations intégrées

## Sécurité

- Azure AD authentication
- Row Level Security (RLS)
- Chiffrement at-rest & in-transit
- VNet integration & Private Endpoints
- Audit logs complets
- GDPR compliant (data purge)

## Opérations

- Auto-scaling (vertical + horizontal)
- Monitoring intégré (Azure Monitor)
- Alertes configurables
- Disaster Recovery (follower databases)
- Continuous backup
- Multi-tenant isolation

## 💡 Quick Start (3 Étapes)

### 1. Créer le Cluster ADX (15 min)

```
az kusto cluster create \
  --cluster-name "loganalysis-prod-adx" \
  --resource-group "rg-loganalysis-prod" \
  --location "westeurope" \
  --sku Standard_E16s_v5 \
  --capacity 8 \
  --enable-streaming-ingest true \
  --zones "1,2,3"
```

### 2. Configurer Ingestion (10 min)

```
# Event Hub
az eventhubs eventhub create \
  --name "application-logs" \
  --namespace-name "loganalysis-prod-eh" \
  --partition-count 32
```

### 3. Créer Table et Ingestion (5 min)

```
.create table ApplicationLogs (
    Timestamp: datetime,
    Level: string,
    Message: string,
    ServiceName: string,
    [...]
)

.create data connection EventHubConnection [...]
```

**Total Time to Value: ~30 minutes ↗**

## 📊 Exemples de Requêtes KQL

### Top Erreurs

```
ApplicationLogs
| where Level == "Error" and Timestamp > ago(24h)
| summarize Count = count() by ErrorType = extract(@"Exception: ([^:]+)", 1,
Exception)
| top 10 by Count desc
```

### Performance P95

```
ApplicationLogs
| where Timestamp > ago(1h)
| summarize P95 = percentile(DurationMs, 95) by ServiceName
| order by P95 desc
```

### Détection Anomalies

```
ApplicationLogs
| make-series ErrorRate = countif(Level == "Error") * 100.0 / count()
    default=0 on Timestamp step 5m
| extend (anomalies, score) = series_decompose_anomalies(ErrorRate, 1.5)
| where anomalies > 0
```

## ⌚ ROI et Bénéfices

### Coûts Évités

Catégorie	Sans ADX	Avec ADX	Économie
<b>Temps debugging</b>	100h/mois	40h/mois	$60h \times \$100 = \$6,000/\text{mois}$
<b>MTTR (incidents)</b>	4 heures	1.5 heures	62% réduction
<b>Incidents évités</b>	-	30% réduction	<b>\$30,000+/mois</b>

ROI: **Positif en 3-6 mois** 

Bénéfices Business

#### Quantifiables:

- ⬇️ 60% temps debugging
- ⬇️ 40% MTTR (Mean Time To Restore)
- ⬇️ 30% incidents production
- ⬆️ 90% query performance vs alternatives

#### Qualitatifs:

- Meilleure satisfaction clients (moins de downtime)
- Insights business (analytics sur comportement users)
- Compliance améliorée (audit trails complets)
- Confiance équipes (visibilité totale)

## Roadmap de Déploiement

### Mois 1-2: Foundation

- ✓ Cluster ADX minimal (6 nœuds)
- ✓ Ingestion batch via Blob
- ✓ Tables et schemas de base
- ✓ KQL queries basiques
- Budget:** \$5,000-7,000/mois

### Mois 3-4: Production

- ✓ Cluster production (8 nœuds)
- ✓ Event Hubs streaming
- ✓ Update policies
- ✓ Dashboards Power BI/Grafana
- Budget:** \$15,000-18,000/mois

### Mois 5-6: Optimisation

- ✓ Reserved Instances (RI 3 ans)
- ✓ Materialized views
- ✓ Tiered storage
- ✓ Auto-scaling
- Budget:** \$13,000-15,000/mois ★

## Mois 7+: Excellence

- Multi-région (DR)
- ML anomaly detection
- Self-service analytics
- Advanced security (RLS)
- Budget:** \$15,000-20,000/mois

## 🔗 Comparaison Alternatives

Solution	Coût/Mois	Pros	Cons
<b>Azure Data Explorer</b>	<b>\$13,465</b>	★ Performance, KQL, Azure natif	Courbe apprentissage KQL
Elasticsearch (AKS)	\$8,000	Mature, flexible	Complexité opérationnelle
Elastic Cloud	\$15,000	Managé	Coût élevé
Splunk Cloud	\$30,000+	Très mature	Très cher
Datadog Logs	\$25,000+	SaaS simple	Très cher à scale
AWS OpenSearch	\$10,000	Managé AWS	Pas Azure natif

**Verdict:** ADX = **Meilleur ratio performance/coût** pour Azure + 5TB/jour

## 📋 Checklist Décision

ADX est le bon choix si:

- Volume > 1 TB/jour
- Besoin queries complexes et rapides
- Infrastructure Azure
- Budget \$10,000-20,000/mois OK
- Équipe peut apprendre KQL (facile)
- Besoin streaming temps réel

Considérer alternatives si:

- Volume < 100 GB/jour (Azure Log Analytics suffit)
- Budget < \$5,000/mois (Elasticsearch self-hosted)
- Déjà investissement lourd Splunk/Elastic
- Queries simples uniquement (grep suffit)
- Pas d'infrastructure Azure

## 📘 Documentation Complète

Document	Description	Pages
<a href="#">README.md</a>	Navigation et quick start	
<a href="#">architecture.md</a>	Architecture globale	

Document	Description	Pages
<a href="#">adx-solution.md</a>	Détails techniques ADX	
<a href="#">kql-examples.md</a>	50+ exemples KQL	
<a href="#">data-ingestion.md</a>	Stratégies ingestion	
<a href="#">cost-estimation.md</a>	Analyse financière détaillée	
<a href="#">diagrams/*.puml</a>	Diagrammes architecture	

## 🎓 Ressources d'Apprentissage

KQL (30 min pour devenir opérationnel)

- [KQL Quick Reference](#)
- [KQL Playground](#)
- [KQL Tutorial](#)

ADX

- [ADX Documentation](#)
- [Best Practices](#)
- [ADX Web UI](#)

Pricing

- [Azure Calculator](#)
- [ADX Pricing](#)

## 🏆 Conclusion

**Pour analyser 5 TB/jour de logs:**

1. **Solution:** Azure Data Explorer avec Event Hubs
2. **Cout:** \$13,465/mois optimisé (RI 3 ans)
3. **Performance:** Sub-seconde queries, <2 min ingestion
4. **ROI:** 3-6 mois
5. **Time to Value:** 30 minutes pour premier cluster

**Recommandation:** ★ GO - Meilleure solution pour ce use case

**Next Steps:**

1.  Review cette documentation
2.  Approuver budget (~\$15K/mois)
3.  Provisionner environnement pilote (\$5K/mois)
4.  Former équipe sur KQL (2-3 jours)
5.  Migration progressive (3-6 mois)

**Questions?** Ouvrir une issue dans ce repository.