Optimizarea Workload-Aware în Cloud-ul Hibrid

ORCHESTRARE SEMANTICĂ ȘI CONFIDENTIAL COMPUTING PENTRU JOBURI SPARK ȘI SERVICII ACTOR

Problema (De ce avem nevoie de acest sistem?)

- Orchestrarea clasică este "**oarbă**": Kubernetes standard alocă resursele bazându-se, în mare parte, pe CPU/RAM disponibil, ignorând specificul procesului (ce face aplicația de fapt).
- **Provocarea Cloud-ului Hibrid:** Avem resurse locale (limitate, dar rapide) și resurse în cloud ("infinite", dar cu latență de rețea și costuri).
- **Nevoia de securitate:** Datele sensibile procesate în cloud public sunt expuse riscurilor la nivel de infrastructură.

Soluția Propusă – Arhitectura Sistemului

- **Conceptul:** Un *Policy-Based Orchestrator* care citește intenția utilizatorului prin metadate (etichete/tag-uri).
- **Fluxul:** Utilizatorul trimite manifestul -> Orchestratorul evaluează tag-urile -> Jobul este direcționat automat către clusterul optim.

Infrastructura		
Cloud Public (Amazon AWS)	putere de calcul brută	scalabilitate
On-Premise (Cluster KinD)	latență minimă	păstrarea datelor local

Demo Workflow

- **Testul 1: Workload cu latență mică (Job Actor)**

Acțiune: Deploy aplicație cu eticheta `type: actor`.

Rezultat: Orchestratorul permite execuția imediată pe clusterul local (KinD).

- **Testul 2: Workload intensiv (Job Spark) – Stadiul de Așteptare**

Acțiune: Deploy aplicație cu eticheta `type: spark`.

Rezultat: Jobul este programat local, dar intră intenționat în starea `Pending` (nu consumă resurse locale limitate).

- **Intervenția Dispatcher-ului (Cloud Migration)**

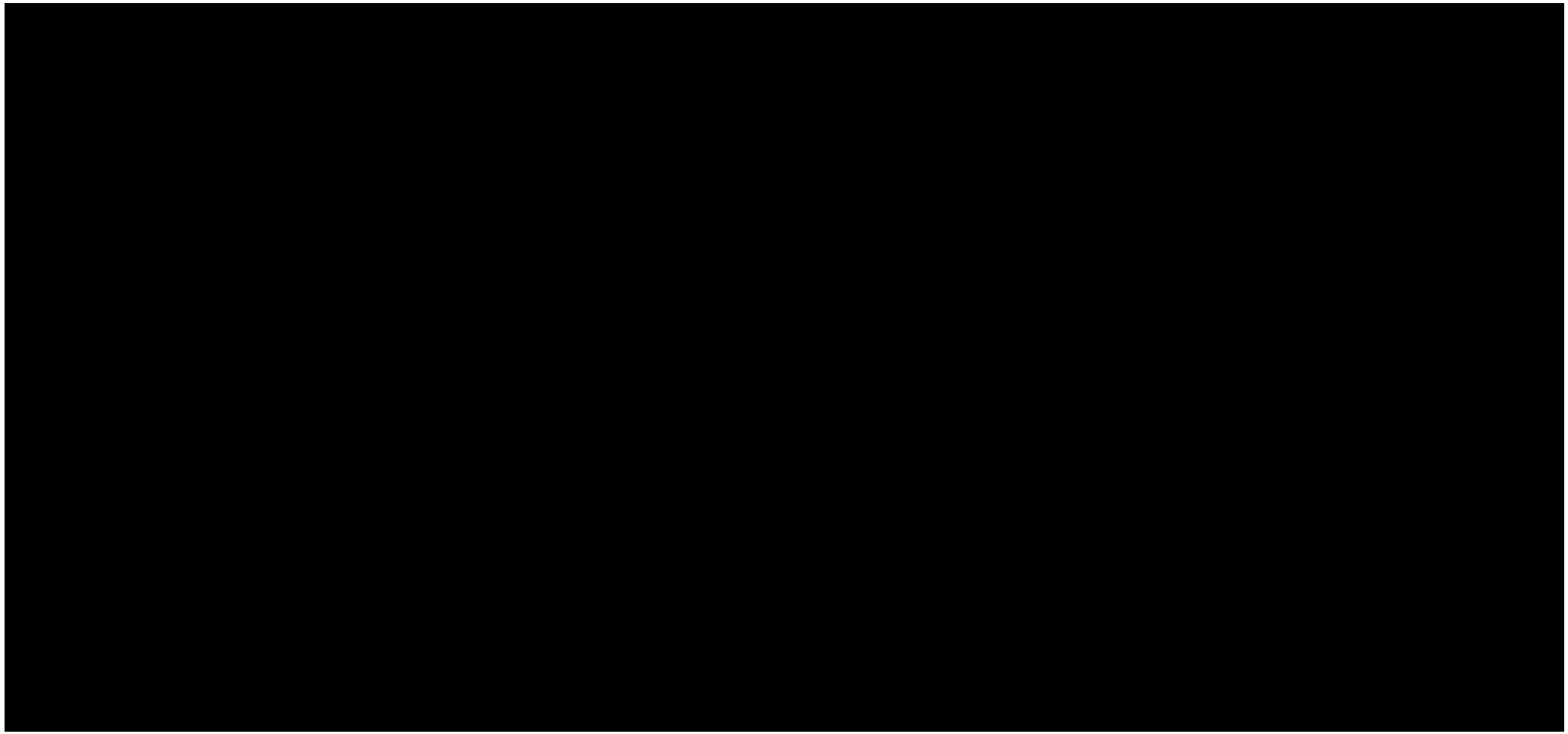
Acțiune: Dispatcher-ul personalizat monitorizează clusterul local.

Rezultat: Detectează pod-urile `Pending` cu eticheta de Spark, se conectează securizat la AWS EKS și le mută pe nodurile din cloud pentru execuție.

Prevenirea Redundanței (Cost Control)

- **Acțiune:** Trimiterea unui job Spark identic cu unul aflat deja în execuție.
- **Rezultat:** Dispatcher-ul detectează duplicatul, blochează migrarea în cloud și emite o alertă pentru a preveni costuri inutile.

Demo



Lecții Învățate

- **Configurare și Securitate AWS (IAM):**

Utilizarea **AWS CLI** pentru interacțiunea programatică cu mediul cloud.

Crearea și gestionarea utilizatorilor prin AWS IAM (Identity and Access Management), alocând corect drepturile de administrator.

- **Provizionarea Infrastructurii Cloud (Amazon EKS):**

Crearea unui cluster Kubernetes administrat de AWS (EKS) și autorizarea utilizatorului de admin pentru accesul la Control Plane.

Configurarea și lansarea de **Node Groups** în cloud pentru a oferi puterea de calcul necesară joburilor Spark.

- **Simularea Mediului On-Premise:**

Configurarea unui cluster local folosind **KinD** (Kubernetes in Docker) pentru a găzdui procesele sensibile la latență (Actorii) și a testa arhitectura hibridă.

- **Securizarea Componentelor Kubernetes (Webhooks):**

Înțelegerea profundă a mecanismelor de extensibilitate K8s.

Generarea de **certificate TLS/SSL** pentru a stabili o conexiune sigură și criptată (HTTPS) între API Server-ul Kubernetes și Webhook-ul personalizat care face rutarea sarcinilor.

- **Tehnici de Depanare (Troubleshooting):**

Utilizarea utilitarelor native (ex. `kubectl describe`, `kubectl logs`, `exec`) pentru investigarea erorilor de execuție ale Pod-urilor și depanarea problemelor de rețea sau de programare (scheduling).

Concluzii și Avantaje

- **Optimizare Semantică (Performanță Maximă):**

Fiecare workload rulează exact unde îi este locul: serviciile Actor beneficiază de latența minimă a rețelei locale, în timp ce joburile Spark folosesc scalabilitatea masivă din Amazon EKS.

- **Controlul Costurilor :**

Prevenirea execuțiilor duplicate prin mecanismul de validare al Dispatcher-ului protejează bugetul de cloud. Infrastructura AWS este consumată doar când este absolut necesar.

- **Securitate by Design (Confidențialitate):**

Sarcinile sensibile rămân izolate pe clusterul local (KinD). Interacțiunea cu cloud-ul este securizată prin certificate TLS/HTTPS și politici stricte IAM (Least Privilege).

- **Automatizare și Abstractizare:**

Dezvoltatorul nu trebuie să cunoască detalii de infrastructură AWS sau Kubernetes. El adaugă doar un simplu tag (etichetă), iar sistemul gestionează complet migrarea, rutarea și validarea.