Extension du Mini Projet : Exceptions, Fichiers, Templates

Imad Kissami

21 Avril 2025

Objectif

- Renforcer le projet cloud avec une gestion robuste des erreurs.
- Ajouter l'exportation des métriques vers un fichier texte.
- Introduire l'utilisation simple de templates pour l'affichage générique.

Étape 8 — Définir les Exceptions Personnalisées

Fichier `CloudExceptions.h`

- Créez une hiérarchie :
  - `CloudException` : hérite de `std::runtime_error`.
  - `AllocationException` : levée lors d'un échec de réservation de ressources.
  - `FileException` : levée si un fichier ne peut être ouvert.

Étape 9 — Intégration dans `Server` et `KubernetesCluster`

- Dans `Server::allocate`, lancer une `AllocationException` si :
  - Le serveur est inactif.
  - Les ressources demandées dépassent ce qui est disponible.

```
throw AllocationException ....
```

- Dans `KubernetesCluster::deployPod`, encapsuler l'appel dans un bloc `try/catch`.

Étape 10 — Gestion de Fichier : Exporter les Métriques

Fichier `Cloud_Util.cpp`

— Ajouter :

```
void saveClusterMetrics(const KubernetesCluster& cluster, const std::string& filename);
```

Cette fonction ouvre un `ofstream`, écrit `getMetrics()` et lève une `FileException` si l'ouverture échoue.

Étape 11 — Gestion des erreurs dans `deployPods`

— Modifier `deployPods` pour les erreurs liées à l'allocation de ressources.
  — Pour chaque pod, si une erreur de type `AllocationException` est levée lors du déploiement, elle est interceptée et un message d'erreur est affiché.
  — Continuer avec les autres pods.
  — À la fin, le vecteur de pods est vidé pour libérer les pointeurs, qu'ils aient été déployés ou non.
— Rajouter ce qu'il faut dans `Cloud_Util.h`.

Étape 12 — Affichage avec `std::setw`

— Adapter les méthodes `getMetrics()` de `Container`, `Pod`, `Server`.
— Utiliser `std::ostringstream`, `std::setw`, `std::left` pour aligner les colonnes.
— Exemple de rendu :

```
[Container: c1     : 2   CPU, 1024  Memory, nginx          ]
[Server: nodeX     | Total: 12   CPU, 12048  MB | Free: 6    CPU, 8976   MB]
```

Étape 13 — Logger Générique avec Template

Fichier `MetricLogger.h`

— Créez une classe template `MetricLogger`.
— Cette fonction écrit `obj.getMetrics()` dans un flux quelconque.
— Utilisation :

```
MetricLogger<KubernetesCluster>::logToStream(cluster, std::cout, "Cluster 1");
```

Étape 14 — Utilisation de Fonctions Lambda

— Ajouter dans `KubernetesCluster` :
  — Une méthode `getFilteredServers` prenant une lambda.

— Dans `Cloud_Util.cpp`, créer :

```
void forEachContainer(const KubernetesCluster& cluster, const std::function<void(const Container&)>& func);
```

— Dans `main.cpp` :
  — Filtrer et afficher les serveurs inactifs avec une lambda.
  — Trier les pods par nombre de conteneurs avec `std::sort` + lambda.
  — Appliquer une lambda à tous les conteneurs pour afficher leurs métriques.

Exemple de main.cpp

```
int main() {
	std::cout << "===␣Test␣AllocationException␣direct␣===\n";
	/* Modifier cette partie pour gérer l'exception*/
	Server failNode("fail-node", 1.0, 1024.0);
	failNode.allocate(4.0, 4096.0); // Trop gros


	std::cout << "\n===␣Test␣FileException␣===\n";
	KubernetesCluster cluster;
	auto nodeX = std::make_shared<Server>("nodeX", 12.0, 12048.0);
	nodeX->start(); // activer le noeud
	cluster.addNode(nodeX); // cluster non vide

	/* Gérer l'erreur d'ouverture du fichier */
	saveClusterMetrics(cluster, "cluster1_metrics.txt");


	std::cout << "\n===␣Test␣Lambda␣:␣serveurs␣inactifs␣===\n";
	KubernetesCluster cluster1;
	auto inactiveServer = std::make_shared<Server>("node3", 2.0, 4096.0); // Ne sera pas activé
	cluster1.addNode(inactiveServer);

	/* Filtrer et afficher les serveurs inactifs à l'aide de la fonction getFilteredServers */
	auto inactifs = cluster1.getFilteredServers([](const Server& s) {
		return !s.isActive();
	});


	std::cout << "\n===␣Déploiement␣sur␣un␣serveur␣inactif␣===\n";

	auto c = std::make_unique<Container>("inactive-c1", "busybox", 1.0, 1024.0);
	auto pod = std::make_unique<Pod>("test-pod");
	pod->addContainer(std::move(c));

	/* Gérer l'erreur ici du deploiement */
	cluster1.deployPod(std::move(pod));



	std::cout << "\n===␣Pods␣triés␣par␣nombre␣de␣conteneurs␣===\n";

	// Création des conteneurs
	auto c1 = std::make_unique<Container>("c1", "nginx", 2.0, 1024.0);
	auto c2 = std::make_unique<Container>("c2", "redis", 4, 2048.0);
	auto c3 = std::make_unique<Container>("c3", "mysql", 2, 1024.0);
	auto c4 = std::make_unique<Container>("c4", "myapp", 10, 12024.0);

	// Création des pods
	auto pod1 = std::make_unique<Pod>("web-pod");
	pod1->addContainer(std::move(c1));
	pod1->addContainer(std::move(c2));
```

```
    auto pod2 = std::make_unique<Pod>("db-pod");
    pod2->addContainer(std::move(c3));

    // Déploiement sans planification réelle, on injecte les pods manuellement
    std::vector<std::unique_ptr<Pod>> pods;
    pods.push_back(std::move(pod1));
    pods.push_back(std::move(pod2));

    /* Gérer le deploiement */
    cluster.deployPod(std::move(pod));



    std::cout << "\n=== Tri des pods ===\n";
    /* Tri des pods */
    std::vector<const Pod*> podRefs;


    std::cout << "\n=== Tous les conteneurs du cluster 1 ===\n";


    return 0;
}
```

Exemple de sortie attendue

```
=== Test AllocationException direct ===
Exception capturée : Allocation Error: Serveur fail-node est inactif

=== Test FileException ===
Métriques sauvegardées avec succès.

=== Test Lambda : serveurs inactifs ===
[Server: node3    | Total: 2    CPU, 4096   MB | Free: 2    CPU, 4096   MB]

=== Déploiement sur un serveur inactif ===
Exception capturée : Allocation Error: Server node3 n'est pas actif

=== Pods triés par nombre de conteneurs ===
 -> Déploiement du Pod [Pod: web-pod]
  [Container: c1    : 2   CPU, 1024  Memory, nginx          ]
  [Container: c2    : 4   CPU, 2048  Memory, redis          ]
 sur le nœud
  [Server: nodeX    | Total: 12   CPU, 12048  MB | Free: 6    CPU, 8976   MB]

 -> Déploiement du Pod [Pod: db-pod]
  [Container: c3    : 2   CPU, 1024  Memory, mysql          ]
 sur le nœud
  [Server: nodeX    | Total: 12   CPU, 12048  MB | Free: 4    CPU, 7952   MB]

=== Tri des pods ===
```

```
[Pod: web-pod]
  [Container: c1    : 2   CPU, 1024  Memory, nginx          ]
  [Container: c2    : 4   CPU, 2048  Memory, redis          ]
[Pod: db-pod]
  [Container: c3    : 2   CPU, 1024  Memory, mysql          ]

=== Tous les conteneurs du cluster 1 ===
  [Container: c1    : 2   CPU, 1024  Memory, nginx          ]
  [Container: c2    : 4   CPU, 2048  Memory, redis          ]
  [Container: c3    : 2   CPU, 1024  Memory, mysql          ]
```

Livrables attendus

- Fichiers :
  - `CloudExceptions.hpp`
  - `MetricLogger.hpp`
  - `Cloud_Util.hpp / .cpp`
  - Modifications dans `KubernetesCluster.hpp / .cpp`
- Tests dans `main.cpp` :
  - Tests d'exceptions
  - Tri et filtrage lambda
  - Affichage formaté
- Fichier généré automatiquement : `cluster1_metrics.txt`