Mini Projet : Modélisation d'une Infrastructure Cloud minimaliste

Imad Kissami

14 Avril 2025

Objectif

- Créer un `Makefile` pour compiler tous les fichiers `.cpp`.
- Tous les fichiers doivent être regroupés dans un dossier `MiniProjet_Nom_Prénom`.
- Simuler une infrastructure cloud avec des serveurs, conteneurs, pods et un cluster Kubernetes.
- Utiliser l'héritage, le polymorphisme et les smart pointers (`std::unique_ptr`, `std::shared_ptr`).
- Les conteneurs ne peuvent être déployés que via un pod, et les pods via un cluster Kubernetes.

Étapes à suivre

Étape 1 — Classe `Resource`

- Classe abstraite servant de base pour toutes les ressources.
- Attributs `protected` :
  - `id_` : identifiant unique (`std::string`).
  - `cpu_` : capacité CPU (`double`).
  - `memory_` : mémoire (`double`).
  - `active_` : état (`bool`).
- Méthodes :
  - `Resource(std::string id, double cpu, double memory);`
    Constructeur.
  - `virtual void start() = 0;`
    Démarre la ressource.
  - `virtual void stop() = 0;`
    Arrête la ressource.

- `virtual std::string getMetrics() const = 0;`
  Retourne les métriques.
- `virtual  Resource() = default;`
  Destructeur virtuel.

Étape 2 — Classe `Container`

- Hérite de : `Resource`.
- Attributs `private` :
  - `image_` : image de l'application (`std::string`).
- Méthodes :
  - `Container(std::string id, std::string image, double cpu, double memory);`
    Constructeur.
  - `void start() override;`
    Active le conteneur.
  - `void stop() override;`
    Désactive le conteneur.
  - `std::string getMetrics() const override;`
    Format : `[Container: id: CPU, Memory, Image]`.
  - `friend std::ostream& operator<<(std::ostream& os, const Container& c);`
    Même format que `getMetrics`.

Étape 3 — Classe `Pod`

- Ne hérite pas de `Resource`, mais gère des conteneurs.
- Attributs `private` :
  - `name_` : nom du pod (`std::string`).
  - `containers_` : liste de `std::unique_ptr<Container>`.
  - `labels_` : métadonnées (`std::unordered_map<std::string, std::string>`).
- Méthodes :
  - `Pod(std::string name, std::unordered_map<std::string, std::string> labels = {});`
    Constructeur.
  - `void addContainer(std::unique_ptr<Container> container);`
    Ajoute un conteneur.
  - `bool removeContainer(const std::string& id);`
    Supprime un conteneur
  - `void deploy();`
    Démarre tous les conteneurs.
  - `std::string getMetrics() const;`
    Agrège les métriques des conteneurs.
  - `friend std::ostream& operator<<(std::ostream& os, const Pod& p);`
    Affiche le nom et les conteneurs.
  - `const std::vector<std::unique_ptr<Container>>& getContainers() const;`
    Récupère les conteneurs

Étape 4 — Classe `Server`

- Hérite de : `Resource`.
- Attributs `private` :
  - `available_cpu_` : CPU disponible (`double`).
  - `available_memory_` : mémoire disponible (`double`).
- Méthodes :
  - `Server(std::string id, double cpu, double memory);`
    Constructeur.
  - `bool allocate(double cpu, double memory);`
    Alloue des ressources si disponibles.
  - `void start() override;`
    Active le serveur.
  - `void stop() override;`
    Désactive le serveur.
  - `std::string getMetrics() const override;`
    Format : `[Server: id: CPU, Memory, Available CPU, Available Memory]`.
  - `friend std::ostream& operator<<(std::ostream& os, const Server& s);`
    Même format que `getMetrics`.

Étape 5 — Classe `KubernetesCluster`

- Gère les serveurs et pods.
- Attributs `private` :
  - `nodes_` : liste de `std::shared_ptr<Server>`.
  - `pods_` : liste de `std::unique_ptr<Pod>`.
- Méthodes :
  - `void addNode(std::shared_ptr<Server> node);`
    Ajoute un serveur.
  - `bool removePod(const std::string& name);`
    Supprimer un Pod
  - `void deployPod(std::unique_ptr<Pod> pod);`
    Déploie un pod sur un serveur disponible.
  - `bool schedulePod(Pod& pod);`
    Tente de planifier un pod sur un serveur disposant de suffisamment de ressources.
  - `Pod* getPod(const std::string& name);`
    Récupère un pod par son nom.
  - `std::string getMetrics() const;`
    Agrège les métriques des serveurs et pods.
  - `friend std::ostream& operator<<(std::ostream& os, const KubernetesCluster& c);`
    Affiche les serveurs et pods.

Étape 6 — Fichier `Cloud_Util.h / .cpp`

— Créez les fonctions suivantes :
  — `void display(const KubernetesCluster& cluster);`
    Affiche les métriques du cluster.
  — `void deployPods(KubernetesCluster& cluster, std::vector<std::unique_ptr<Pod>>& pods);`
    Déploie plusieurs pods.

Étape 7 — Fichier `main.cpp`

```
int main() {
      std::cout.precision(2);
      std::cout << std::fixed;

      std::cout << "\n=== Cluster 1 ===\n";

      // Créer un vecteur de clusters
      std::vector<KubernetesCluster> clusters(2); // Deux clusters

      // === Configurer le premier cluster ===
      KubernetesCluster& cluster1 = clusters[0];
      cluster1.addNode(std::make_shared<Server>("node1-1", 4.0, 8192.0));
      cluster1.addNode(std::make_shared<Server>("node1-2", 8.0, 14096.0));

      // Créer des conteneurs pour le premier cluster
      auto c1_1 = std::make_unique<Container>("c1-1", "nginx:latest", 2, 5120.0);
      auto c1_2 = std::make_unique<Container>("c1-2", "redis:latest", 6, 2560.0);

      // Créer un pod pour le premier cluster
      auto pod1_1 = std::make_unique<Pod>("web-pod", std::unordered_map<std::string,
       std::string>{{"app", "nginx"}});
      pod1_1->addContainer(std::move(c1_1));
      pod1_1->addContainer(std::move(c1_2));


      // Créer un autre pod pour le premier cluster
      auto c1_3 = std::make_unique<Container>("c1-3", "mysql:latest", 0.7, 1024.0);
      auto pod1_2 = std::make_unique<Pod>("db-pod", std::unordered_map<std::string,
       std::string>{{"app", "mysql"}});
      pod1_2->addContainer(std::move(c1_3));


      // Déployer les pods dans le premier cluster
      std::vector<std::unique_ptr<Pod>> pods1;
      pods1.push_back(std::move(pod1_1));
      pods1.push_back(std::move(pod1_2));
      deployPods(cluster1, pods1);

      // === Test des fonctionnalités ===

      // Test 0 : Déploiement 'dun pod avec ressources insuffisantes
      std::cout << "\n=== Test 0 : Déploiement 'dun pod avec ressources insuffisantes ===\n";
      auto large_container = std::make_unique<Container>("large-c1", "large-app:latest", 5.0, 10000.0);
      auto large_pod = std::make_unique<Pod>("large-pod", std::unordered_map<std::string,
       std::string>{{"app", "large"}});
      large_pod->addContainer(std::move(large_container));
      cluster1.deployPod(std::move(large_pod));


      std::cout << "\n=== Cluster 2 ===\n";

      // === Configurer le second cluster ===
```

```
KubernetesCluster& cluster2 = clusters[1];
cluster2.addNode(std::make_shared<Server>("node2-1", 3.0, 6144.0));
cluster2.addNode(std::make_shared<Server>("node2-2", 1.5, 2048.0));

// Créer des conteneurs pour le second cluster
auto c2_1 = std::make_unique<Container>("c2-1", "node:latest", 0.6, 768.0);
auto c2_2 = std::make_unique<Container>("c2-2", "memcached:latest", 0.4, 384.0);

// Créer un pod pour le second cluster
auto pod2_1 = std::make_unique<Pod>("api-pod", std::unordered_map<std::string,
 std::string>{{"app", "node"}});
pod2_1->addContainer(std::move(c2_1));

// Créer un autre pod pour le second cluster
auto c2_2_2 = std::make_unique<Container>("c2-2-2", "memcached:latest", 0.4, 384.0);
auto pod2_2 = std::make_unique<Pod>("cache-pod", std::unordered_map<std::string,
 std::string>{{"app", "memcached"}});
pod2_2->addContainer(std::move(c2_2_2));

// Déployer les pods dans le second cluster
std::vector<std::unique_ptr<Pod>> pods2;
pods2.push_back(std::move(pod2_1));
pods2.push_back(std::move(pod2_2));
deployPods(cluster2, pods2);


// Afficher 'létat initial
std::cout << "\n=== État initial ===\n";
for (size_t i = 0; i < clusters.size(); ++i) {
       std::cout << "\n=== Metrics for Cluster " << (i + 1) << " ===\n";
       display(clusters[i]);
}

// Test 1 : Supprimer un conteneur (c1-2 du web-pod dans cluster1)
std::cout << "\n=== Test 1 : Suppression du conteneur c1-2 (redis) ===\n";
Pod* web_pod = clusters[0].getPod("web-pod");
if (web_pod && web_pod->removeContainer("c1-2")) {
       std::cout << "Conteneur c1-2 supprimé du web-pod\n";
} else {
       std::cout << "Conteneur c1-2 ou web-pod non trouvé\n";
}

// Afficher 'létat après suppression du conteneur
std::cout << "\n=== État après suppression du conteneur ===\n";
for (size_t i = 0; i < clusters.size(); ++i) {
       std::cout << "\n=== Metrics for Cluster " << (i + 1) << " ===\n";
       display(clusters[i]);
}

// Test 2 : Supprimer un pod (db-pod du cluster1)
std::cout << "\n=== Test 2 : Suppression du pod db-pod ===\n";
if (clusters[0].removePod("db-pod")) {
       std::cout << "Pod db-pod supprimé du cluster 1\n";
} else {
       std::cout << "Pod db-pod non trouvé\n";
}

// Afficher 'létat après suppression du pod
std::cout << "\n=== État après suppression du pod ===\n";
for (size_t i = 0; i < clusters.size(); ++i) {
       std::cout << "\n=== Metrics for Cluster " << (i + 1) << " ===\n";
       display(clusters[i]);
}

// Test 3 : Supprimer un cluster (cluster2)
std::cout << "\n=== Test 3 : Suppression du cluster 2 ===\n";
clusters.erase(clusters.begin() + 1);
```

```
        // Afficher l'état final
        std::cout << "\n=== État après suppression du cluster ===\n";
        for (size_t i = 0; i < clusters.size(); ++i) {
                std::cout << "\n=== Metrics for Cluster " << (i + 1) << " ===\n";
                display(clusters[i]);
        }

        return 0;
}
```

Tests recommandés

- Déploiement d'un pod sur un serveur sans ressources suffisantes.
- Ajout de plusieurs conteneurs à un pod.
- Affichage des métriques pour un cluster vide.
- Déploiement de plusieurs pods sur différents serveurs.
- Vérification que `start()` et `stop()` modifient l'état.

Output attendu :

```
=== Cluster 1 ===
-> Déploiement du Pod [Pod: web-pod]
[Container: c1-1: 2 CPU, 5120 Memory, nginx:latest]
[Container: c1-2: 6 CPU, 2560 Memory, redis:latest]
sur le nœud [Server: node1-2: 8 CPU, 14096 Memory, Available: 0 CPU, 6416 Memory]
Pod [Pod: web-pod]
[Container: c1-1: 2 CPU, 5120 Memory, nginx:latest]
[Container: c1-2: 6 CPU, 2560 Memory, redis:latest]
déployé avec succès.
-> Déploiement du Pod [Pod: db-pod]
[Container: c1-3: 0.7 CPU, 1024 Memory, mysql:latest]
sur le nœud [Server: node1-1: 4 CPU, 8192 Memory, Available: 3.3 CPU, 7168 Memory]
Pod [Pod: db-pod]
[Container: c1-3: 0.7 CPU, 1024 Memory, mysql:latest]
déployé avec succès.

=== Test 0 : Déploiement d'un pod avec ressources insuffisantes ===
Échec du déploiement du pod [Pod: large-pod]
[Container: large-c1: 5 CPU, 10000 Memory, large-app:latest]
: ressources insuffisantes.

=== Cluster 2 ===
-> Déploiement du Pod [Pod: api-pod]
[Container: c2-1: 0.6 CPU, 768 Memory, node:latest]
sur le nœud [Server: node2-1: 3 CPU, 6144 Memory, Available: 2.4 CPU, 5376 Memory]
Pod [Pod: api-pod]
```

```
[Container: c2-1: 0.6 CPU, 768 Memory, node:latest]
déployé avec succès.
-> Déploiement du Pod [Pod: cache-pod]
[Container: c2-2-2: 0.4 CPU, 384 Memory, memcached:latest]
sur le nœud [Server: node2-1: 3 CPU, 6144 Memory, Available: 2 CPU, 4992 Memory]
Pod [Pod: cache-pod]
[Container: c2-2-2: 0.4 CPU, 384 Memory, memcached:latest]
déployé avec succès.

=== État initial ===

=== Metrics for Cluster 1 ===
=== Cluster Metrics ===
[Server: node1-1: 4 CPU, 8192 Memory, Available: 3.3 CPU, 7168 Memory]
[Server: node1-2: 8 CPU, 14096 Memory, Available: 0 CPU, 6416 Memory]
[Pod: web-pod]
[Container: c1-1: 2 CPU, 5120 Memory, nginx:latest]
[Container: c1-2: 6 CPU, 2560 Memory, redis:latest]
[Pod: db-pod]
[Container: c1-3: 0.7 CPU, 1024 Memory, mysql:latest]

=== Metrics for Cluster 2 ===
=== Cluster Metrics ===
[Server: node2-1: 3 CPU, 6144 Memory, Available: 2 CPU, 4992 Memory]
[Server: node2-2: 1.5 CPU, 2048 Memory, Available: 1.5 CPU, 2048 Memory]
[Pod: api-pod]
[Container: c2-1: 0.6 CPU, 768 Memory, node:latest]
[Pod: cache-pod]
[Container: c2-2-2: 0.4 CPU, 384 Memory, memcached:latest]

=== Test 1 : Suppression du conteneur c1-2 (redis) ===
Conteneur c1-2 supprimé du web-pod

=== État après suppression du conteneur ===

=== Metrics for Cluster 1 ===
=== Cluster Metrics ===
[Server: node1-1: 4 CPU, 8192 Memory, Available: 3.3 CPU, 7168 Memory]
[Server: node1-2: 8 CPU, 14096 Memory, Available: 0 CPU, 6416 Memory]
[Pod: web-pod]
[Container: c1-1: 2 CPU, 5120 Memory, nginx:latest]
[Pod: db-pod]
[Container: c1-3: 0.7 CPU, 1024 Memory, mysql:latest]

=== Metrics for Cluster 2 ===
```

```
=== Cluster Metrics ===
[Server: node2-1: 3 CPU, 6144 Memory, Available: 2 CPU, 4992 Memory]
[Server: node2-2: 1.5 CPU, 2048 Memory, Available: 1.5 CPU, 2048 Memory]
[Pod: api-pod]
[Container: c2-1: 0.6 CPU, 768 Memory, node:latest]
[Pod: cache-pod]
[Container: c2-2-2: 0.4 CPU, 384 Memory, memcached:latest]

=== Test 2 : Suppression du pod db-pod ===
Pod db-pod supprimé du cluster 1

=== État après suppression du pod ===

=== Metrics for Cluster 1 ===
=== Cluster Metrics ===
[Server: node1-1: 4 CPU, 8192 Memory, Available: 3.3 CPU, 7168 Memory]
[Server: node1-2: 8 CPU, 14096 Memory, Available: 0 CPU, 6416 Memory]
[Pod: web-pod]
[Container: c1-1: 2 CPU, 5120 Memory, nginx:latest]

=== Metrics for Cluster 2 ===
=== Cluster Metrics ===
[Server: node2-1: 3 CPU, 6144 Memory, Available: 2 CPU, 4992 Memory]
[Server: node2-2: 1.5 CPU, 2048 Memory, Available: 1.5 CPU, 2048 Memory]
[Pod: api-pod]
[Container: c2-1: 0.6 CPU, 768 Memory, node:latest]
[Pod: cache-pod]
[Container: c2-2-2: 0.4 CPU, 384 Memory, memcached:latest]

=== Test 3 : Suppression du cluster 2 ===

=== État après suppression du cluster ===

=== Metrics for Cluster 1 ===
=== Cluster Metrics ===
[Server: node1-1: 4 CPU, 8192 Memory, Available: 3.3 CPU, 7168 Memory]
[Server: node1-2: 8 CPU, 14096 Memory, Available: 0 CPU, 6416 Memory]
[Pod: web-pod]
[Container: c1-1: 2 CPU, 5120 Memory, nginx:latest]
```