وظيفة مقرر اساسيات النظم الموزعة

- ضمن هذا المشروع لدينا ما يلي:
  - Registration
  - Discovery
  - Leader reelection
  - Auto healing
- ضمن هذا التقرير سنقوم فقط بشرح قسم auto healing الذي سيكون من مسؤولية القائد الذي سيتم انتخابه:
  - عندما يتم انتخاب قائد سيقوم هذا القائد بإلغاء تسجيله من ال cluster.
  - سيقوم القائد باستكشاف العقد الموجودة ضمن ال cluster.
  - ومن ثم سيقوم بتشغيل n نسخة من التطبيق المراد تشغيله على هذه العقد، أي أننا بحاجة خوارزمية جدولة لتقسيم الحمل على العقد بالتساوي.
  - عندما تتوقف أحد نسخ التطبيق عن العمل على أحد العقد ستكون مسؤولية القائد اكتشاف ان أحد النسخ توقفت وعليه أن يقوم بتشغيل نسخة بديلة مع مراعاة توزيع الحمل على أحد العقد.
  - عندما تتوقف عقدة كاملة عن العمل سيكون من مسؤولية القائد اكتشاف توقف العقد، وتشغيل نسخ من التطبيق بدلاً من النسخ التي كانت تعمل على العقدة التي توقفت.

- **طريقة تنجيز ما سبق كما يلي:**
  - أولا جميع عقد ال cluster ستسجل نفسها تحت service_registry/، حيث سيتم انشاء znode خاصة بكل عقدة وتكون تحتوي عنوان العقدة، وسيتم تخزين العنوان على شكل user_name@ip_address.
  - لذلك يستطيع القائد معرفة جميع عقد ال cluster من خلال التعليمة getChildren.
  - ومن خلال تعليمة getData لكل znode يستطيع معرفة عناوين العقد.
  - سيقوم القائد بتقسيم المهام على العقد بالتساوي وفق خوارزمية جدولة بسيطة سنقوم بشرحها لاحقاً.
  - عندما يتم تحديد عدد النسخ الواجب تشغيله على كل عقدة سيقوم القائد بفتح اتصال ssh مع العقد وتشغيل النسخ عليها.
  - عند تشغيل نسخة من التطبيق على أحد العقد سيتم انشاء znode تحت workers/ وتكون تحتوي عنوان العقدة االتي قامت التي قامت بإنشاءها.
  - عندما تتوقف أحد النسخ عن العمل سيتم حذف ال znode الموافقة.
  - أيضا من خلال تعليمة getChildren(/workers) يستطيع القائد معرفة عدد النسخ التي تعمل حاليا.
  - أي أن القائد أصبح قادراً على معرفة عدد العقد التي تعمل حالياً، وعدد النسخ التي تعمل حالياً، مع العلم إنه لديه عدد النسخ الثابت الواجب تشغيله على جميع العقد.

- القائد سيراقب التغيرات التي تحصل على workers/ لمعرفة ما اذا تم انشاء نسخة جديدة من التطبيق او تم توقف نسخة.

- **خوارزمية الجدولة وتقسيم الحمل:**

- بدايةً نقوم بتعريف بنية معطيات للربط بين string يمثل عنوان عقدة وinteger يمثل عدد النسخ التي تعمل حالياً على هذه العقدة (nodesCountMap).

```
private Map<String, Integer> nodesCountMap = new HashMap<>();
```

- يقوم القائد بالحصول على جميع أبناء workers/ لمعرفة عددهم.

```
List<String> workerZnodes = zooKeeper.getChildren(AUTOHEALER_ZNODES_PATH, watcher: this);
int existingWorkers = workerZnodes.size();
```

- نقوم بالمرور على جميع أبناء workers/ والحصول على المعطيات المخزّنة ضمنها ومن خلالها نعرف عنوان العقدة التي أنشئت هذه النسخة، وبناءً عليه نقوم بتخزين عنوان العقد التي تعمل حالياً مع عدد النسخ التي تعمل عليها.

```
for (String child : workerZnodes) {
    String znodePath = AUTOHEALER_ZNODES_PATH + "/" + child;
    byte[] data = zooKeeper.getData(znodePath, watch: false, stat: null);
    String workerAddress = new String(data);
    nodesCountMap.put(workerAddress, nodesCountMap.getOrDefault(workerAddress, defaultValue: 0) + 1);
}
```

- وأيضاً يجب الانتباه إلى انه في هذه الأثناء ربما يوجد عقد جديدة قد بدأت بالعمل لذلك يجب علينا أن نسند إليها مهام لتنفيذها، ويمكن معرفة هذا من خلال الحصول على أبناء service_registery/.

```
List<String> allServiceAddresses = serviceRegistry.getAllServiceAddresses();
```

- وسنقوم بإضافة عناوين العقد الجديدة إلى nodesCountMap مع عدد نسخ يساوي الصفر.

```
for (String address : allServiceAddresses) {
    if (!nodesCountMap.containsKey(address)) {
        nodesCountMap.put(address, 0);
    }
}
```

- عدد العقد الإجمالي للعقد يساوي عدد أبناء service_registery/.

```
int numNodes = allServiceAddresses.size();
```

- تقوم خوارزمية الجدولة البسيطة على ما يلي تقسيم عدد النسخ الواجب تشغيلها على عدد العقد الكلي، وبهذا نعلم ما هو عدد العقد الواجب تشغيله على كل عقدة.

```
int averageWorkersPerNode = 0;
int remainingWorkers = 0;
if(numNodes != 0){
    averageWorkersPerNode= numberOfWorkers / numNodes;
    remainingWorkers = numberOfWorkers % numNodes;
}
```

- الأن بعد أن علمنا عدد النسخ الكلي m الواجب تشغيله على كل عقدة يجب أن نقوم بحساب ما هو عدد النسخ الواجب تشغيله على كل عقدة حتى نحصل على m نسخة، وهذا العدد يساوي متوسط النسخ الواجب تسغيله على كل عقدة مطروحاً منه عدد النسخ التي تعمل حالياً على العقدة.

```
Map<String, Integer> workersToAdd = new HashMap<>();
for (String address : allServiceAddresses) {
    int currentWorkers = nodesCountMap.getOrDefault(address, defaultValue: 0);
    int workersNeeded = Math.max(0, averageWorkersPerNode - currentWorkers);
    workersToAdd.put(address, workersNeeded);
}
```

- وأيضاً يجب أن نقوم بمعالجة حالة أن العدد الإجمالي لا يقبل القسمة بشكل صحيح على عدد العقد.

```
List<String> sortedNodes = new ArrayList<>(allServiceAddresses);
sortedNodes.sort(Comparator.comparingInt(node -> nodesCountMap.getOrDefault(node, defaultValue: 0)));
int nodeIndex = 0;
for (int i = 0; i < remainingWorkers; i++) {
    String nodeAddress = sortedNodes.get(nodeIndex);
    workersToAdd.put(nodeAddress, workersToAdd.get(nodeAddress) + 1);
    nodeIndex = (nodeIndex + 1) % numNodes;
}
```

- ومن ثم نقوم بحساب العدد الإجمالي للنسخ الواجب تشغيله workerTolaunch حتى نحصل على n نسخة تعمل على كامل عقد ال cluster.

```
int workersToLaunch = 0;
if (existingWorkers < numberOfWorkers) {
    workersToLaunch = numberOfWorkers - existingWorkers;
}
```

- ثم نبدأ بتشغيل النسخ على العقد حتى نحصل يصبح عدد النسخ التي تم تشغيلها يساوي workerTolaunch.

```
for (String address : sortedNodes) {
    int myWorkersToLaunch = workersToAdd.get(address);
    for (int i = 0; i < myWorkersToLaunch && workersToLaunch> 0; i++) {
        try {
            startNewWorker(address, pathToProgram);
            workersToLaunch--;
            nodesCountMap.put(address, nodesCountMap.getOrDefault(address, defaultValue: 0) + 1);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```

- في النهاية ولكون عملية تشغيل النسخ تحتاج إلى وقت، وعندما يتم تشغيل عدة نسخ معاً ستبدأ العمل بعد فترات متفاوتة، وهناك نسخ أخرى يمكن أن تتوقف في أي لحظة وجميع ما سبق يوّلد أحداث تجعل القائد يعيد تنفيذ خوارزمية الجدولة وتوزيع الحمل لذلك يجب أن أن يقوم القائد بعد كل تنفيذ لخوازمية الجدولة وتقسيم الحمل بالإنتظار قليلاً تكون جميع النسخ التي قام بتشغيلها على العقد الأخرى قد بدأت العمل.

```
Thread.sleep( millis: 30000);
```

مثال:

على فرض أننا نريد تشغيل 8 نسخ من التطبيق على كامل العقد، وفي لحظة معينة توقفت بعض النسخ عن العمل وأصبح العدد الإجمالي هو 4، سيقوم القائد بمعالجة هذه المشكلة وسيقوم بتشغيل 4 نسخ أخرى ولكون هذه ستبدأ العمل في أوقات مختلفة، عندما تعمل أول نسخة، مباشرة سيلاحظ القائد التغير وسيتفقد ما اذا كان هناك نقص في عدد النسخ الواجب تشغيله على كامل ال cluster وبالفعل سيجد أن العدد الإجمالي هو 5 وليس 8 لأن النسخ الثلاثة الباقية لم تبدأ العمل بعد، وسيقوم بتشغيل ثلاثة نسخ أخرى وهكذا وبهذا الشكل سيصبح العدد الإجمالي لا يساوي العدد المطلوب (8). لذلك نجعل القائد ينتظر قليلاً حتى نضمن أن جميع النسخ قد بدأت العمل، أي أن القائد سيقوم بالمعافاة التلقائية لل cluster كل 30 ثانية.

ملاحظات:

- عملية اتصال ssh من الألة الإفتراضية إلى الألة الحقيقة فشلت، لذلك سيتم تشغيل المشروع اعتماداً على أن الألة الحقيقية ستلعب دور القائد، أي أنها أول ألة سيتم تشغيلها ضمن ال cluster.
- بسبب الأمر السابق ولكون الألات الأخرى جميعها ubuntu وتحتوي نفس المسار تم كتابة المسار إلى ملف ال jar الذي يمثل worker program بشكل ثابت ضمن تابع startNewWorker.

الطالب: علي حسان سعيد.