

.so.b@k@

Java
cloud
computing
AOP
framework

N*Новосибирский
государственный
университет
***НАСТОЯЩАЯ НАУКА**

Выполнил:
Смоляков П.Е.

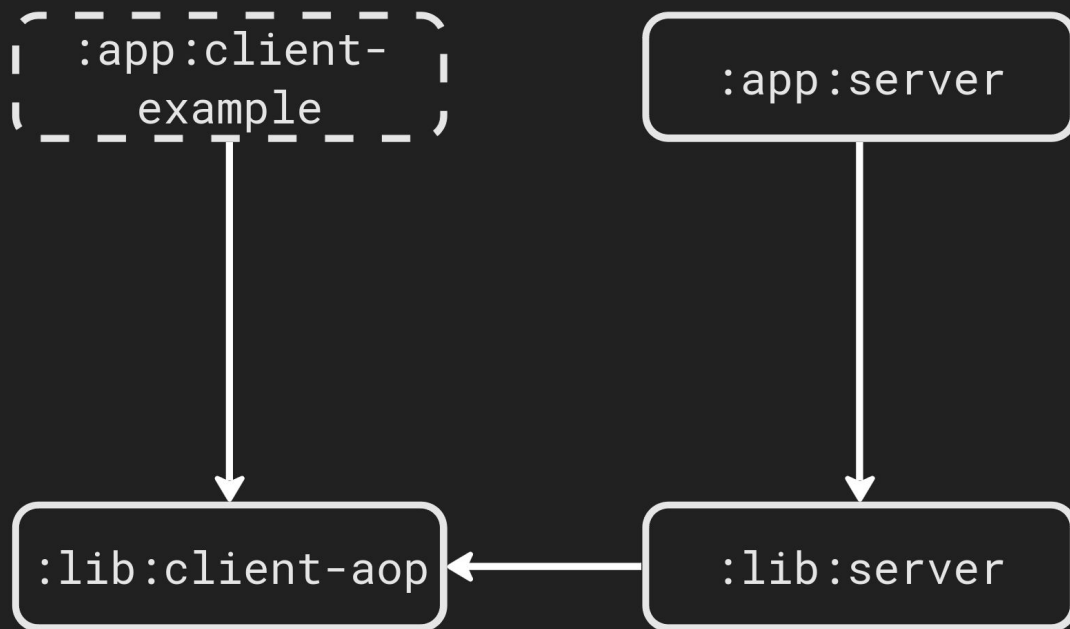
О чём проект

- Нередко возникает необходимость осуществлять долгие вычисления на более производительном удалённом сервере/кластере
- Хотелось бы иметь интегрированный в язык программирования инструмент, предлагающий возможности облачного вычисления без ручной работы по передаче кода и данных

Требования по реализации

- Фреймворк для Java
- Доступно выполнение произвольного кода
- Все взаимодействие с сервером/кластером должно выполняться без участия пользователя; он должен лишь сообщить о намерении вычислять метод удалённо
- Допустимо существование альтернативных реализаций клиентской части фреймворка; таким образом, серверная часть должна обладать удобным универсальным API
- Хотелось бы безопасно

Структура проекта



Как использовать

```
@SobakaCloudCompute(server = "pentium3.com")
public class ClassToCalculate {
    @SobakaEntryMethod
    public static int[] calculateMe(int[] arr) {
        // CODE HERE
        return null;
    }
}
```

Класс, содержащий методы с вычислениями, и **статический** метод - точка старта вычислений - отмечаются соответствующими аннотациями.

Статический, чтобы не передавать еще и состояние класса.

Как использовать

Фреймворком создаётся класс со статическим методом, запускающим облачные вычисления. Исходный класс тоже можно использовать по необходимости.

```
public static void main(String[] args) {  
    var arr = ClassToCalculateSobakaCloud.calculateMe(  
        new int[]{1, 2, 3, 4, 5}  
    );  
    System.err.println(Arrays.toString(arr));  
}
```

Подробнее про собак

```
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.TYPE)
public @interface SobakaCloudCompute {
    String server();

    String targetShortClassName() default
        CloudComputeProcessor.EMPTY_VALUE;

    String targetPackage() default
        CloudComputeProcessor.EMPTY_VALUE;
}
```

Подробнее про собак

```
@Retention(RetentionPolicy.SOURCE)
@Target(ElementType.METHOD)
public @interface SobakaEntryMethod {
    String targetName() default
        CloudComputeProcessor.EMPTY_VALUE;

    int pollingIntervalMillis() default
        HttpClientCloudComputingClient.DEFAULT_POLLING_INTERVAL_MILLIS;

    int sleepBeforePollingMillis() default
        HttpClientCloudComputingClient.DEFAULT_SLEEP_BEFORE_POLLING_MILLIS;
}
```


Подробнее про собак

Вместо инструментации предлагается использовать кодогенерацию времени компиляции.

- Выявление некорректного использования фреймворка во время компиляции
- В том числе, проверка типов и правильной расстановки аннотаций во время компиляции

Кодогенерация выполняется стандартными механизмами Java (см. [AbstractProcessor](#)).

Пример сгенерированного класса

```
package ru.nsu.fit.smolyakov.sobakacloud.aop.appexample;

public class ClassToCalculateSobakaCloud {
    public static int[] calculateMe(int[] arr) {
        return (int[]) ru.nsu.fit.smolyakov.sobakacloud.aop.HttpCloudComputingClient.send(
            ru.nsu.fit.smolyakov.sobakacloud.aop.appexample.ClassToCalculate.class,
            "calculateMe",
            java.util.List.of(int[].class),
            java.util.List.of(arr),
            int[].class,
            0, 1000
        );
    }
} // каюсь, табы проставлены после, как и этот комментарий
```

Клиент

```
public static Object send(Class<?> clazz,  
                          String methodName,  
                          List<Class<?>> argTypes,  
                          List<Object> args,  
                          Class<?> retType,  
                          long sleepBeforePollingMillis,  
                          long pollingIntervalMillis) {  
    ...  
}
```

Клиент

- 1) Откуда-то достаём байткод переданного класса. Таким образом, вместе с основным методом мы еще и передадим вложенные (если они из того же класса).
- 2) Сериализуем аргументы метода как массив пар *тип-значение*, сохраняя их порядок. На данный момент поддерживаются строки, примитивы и массивы примитивов.
- 3) Отправляем серверу по HTTP.

В качестве HTTP клиента (и сервера тоже) используется Jetty 11.

Где взять байткод класса

- Изначально была идея перехватывать с помощью Instrumentation API байтовые представления классов до того, как они попали в `defineClass` класслоадера
- Однако почему бы просто не прочитать `.class` файл?
- Работает в том числе с `.jar`-ами
- Не работает с кодом, изменившимся уже в рантайме

```
try (var stream = Thread.currentThread()  
    .getContextClassLoader().getResourceAsStream(ClassToCalculate.class)) {  
    return Objects.requireNonNull(stream).readAllBytes();  
}
```

Сериализуем аргументы

- Можно было бы упаковывать аргументы более компактно в каком-то бинарном виде
- Однако предоставить человекочитаемый REST API в данном случае кажется предпочтительнее, поскольку серверной частью сможет пользоваться даже не Java приложение
- Например, клиент на условном Python может отсылать серверу предварительно скомпилированный байткод и какие-то свои аргументы
- Такой способ, впрочем, не блещет эффективностью по памяти и производительностью
- json'ы (де-)сериализуем библиотекой jackson

Сериализуем аргументы

```
public enum Type {  
    @JsonProperty("byte") BYTE(byte.class, name: "byte", (Object value) -> new ByteArgDto((byte) value)),  
    @JsonProperty("short") SHORT(short.class, name: "short", (Object value) -> new ShortArgDto((short) value)),  
    @JsonProperty("int") INT(int.class, name: "int", (Object value) -> new IntArgDto((int) value)),  
    @JsonProperty("long") LONG(long.class, name: "long", (Object value) -> new LongArgDto((long) value)),  
    @JsonProperty("float") FLOAT(float.class, name: "float", (Object value) -> new FloatArgDto((float) value)),  
    @JsonProperty("double") DOUBLE(double.class, name: "double", (Object value) -> new DoubleArgDto((double) value)),  
    @JsonProperty("boolean") BOOLEAN(boolean.class, name: "boolean", (Object value) -> new BooleanArgDto((boolean) value)),  
    @JsonProperty("char") CHAR(char.class, name: "char", (Object value) -> new CharArgDto((char) value)),  
  
    @JsonProperty("byte[]") BYTE_ARRAY(byte[].class, name: "byte[]", (Object value) -> new ByteArrayArgDto((byte[]) value)),  
    @JsonProperty("short[]") SHORT_ARRAY(short[].class, name: "short[]", (Object value) -> new ShortArrayArgDto((short[]) value)),  
    @JsonProperty("int[]") INT_ARRAY(int[].class, name: "int[]", (Object value) -> new IntArrayArgDto((int[]) value)),  
    @JsonProperty("long[]") LONG_ARRAY(long[].class, name: "long[]", (Object value) -> new LongArrayArgDto((long[]) value)),  
    @JsonProperty("float[]") FLOAT_ARRAY(float[].class, name: "float[]", (Object value) -> new FloatArrayArgDto((float[]) value)),  
    @JsonProperty("double[]") DOUBLE_ARRAY(double[].class, name: "double[]", (Object value) -> new DoubleArrayArgDto((double[]) value)),  
    @JsonProperty("boolean[]") BOOLEAN_ARRAY(boolean[].class, name: "boolean[]", (Object value) -> new BooleanArrayArgDto((boolean[]) value)),  
    @JsonProperty("char[]") CHAR_ARRAY(char[].class, name: "char[]", (Object value) -> new CharArrayArgDto((char[]) value)),  
  
    @JsonProperty("java.lang.String") STRING(String.class, name: "java.lang.String", (Object value) -> new StringArgDto((String) value));  
}
```

Гоняем json'ы

```
POST /compute/submit HTTP/1.1
```

```
Content-Type: multipart/form-data; boundary=SobakaBoundary
```

```
--SobakaBoundary
```

```
Content-Disposition: form-data; name="classFile"
```

```
<<bytecode>>
```

```
--SobakaBoundary
```

```
Content-Disposition: form-data; name="requestInfo"
```

```
{"entryMethodName": "calculateMe",  
  "args": [ {"argValue": [1,2,3,4,5], "argType": "int[]"} ]}
```

```
--SobakaBoundary--
```

```
resp: id
```


Что сервер отправляет обратно

Получает: айди

Возвращает:

- `inProgress`: задача в данный момент считается
- `success`: задача посчиталась, получаем результат
- `failure`: задача прервалась из-за исключения. Создаётся экземпляр класса `SobakaExecutionException (unchecked)` с информацией об исходном исключении. После десериализации и передачи клиенту выбрасывается в теле метода `send`.

Угоняем json'ы

GET /compute/result?id=100500

resp:

```
{ "status": "success", "argDto": {  
  "argValue": [6,7,8,9,10],  
  "argType": "int[]" } }
```

resp:

```
{ "status": "failure", "exception":  
  { "exceptionMsg": "java.lang.NullPointerException: npe example" } }
```

resp:

```
{ "status": "inProgress" }
```

Подробнее про сервер

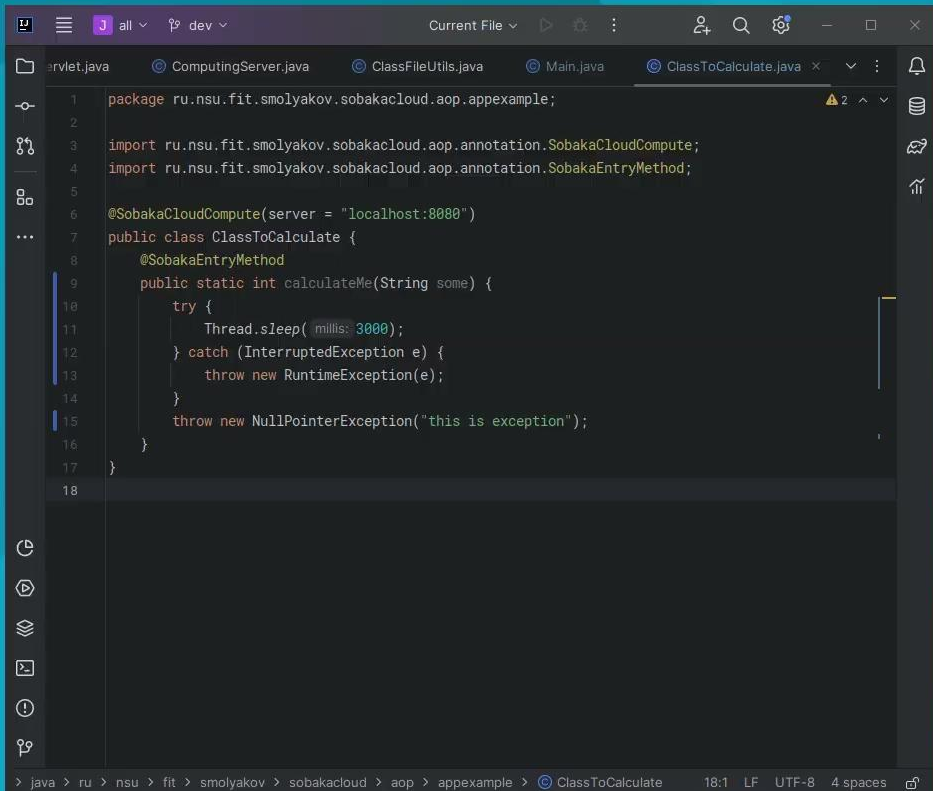
- Сервер получает и парсит запрос на submit
- Десериализует аргументы
- Загружает класс через кастомный ClassLoader
- Отправляет все в класс-обертку над SingleThreadPoolExecutor (треды, если нужны, можно создать самостоятельно)
- Потом отвечает в result'ах так, как это было описано два слайда назад

Кастомный ClassLoader

- Незамысловатое решение

```
public class BytesClassLoader extends SecureClassLoader {  
    public BytesClassLoader() {  
    }  
  
    public Class<?> loadClassFromBytes(byte[] bytes) {  
        return defineClass(  
            null, bytes, 0, bytes.length, (CodeSource) null  
        );  
    }  
}
```

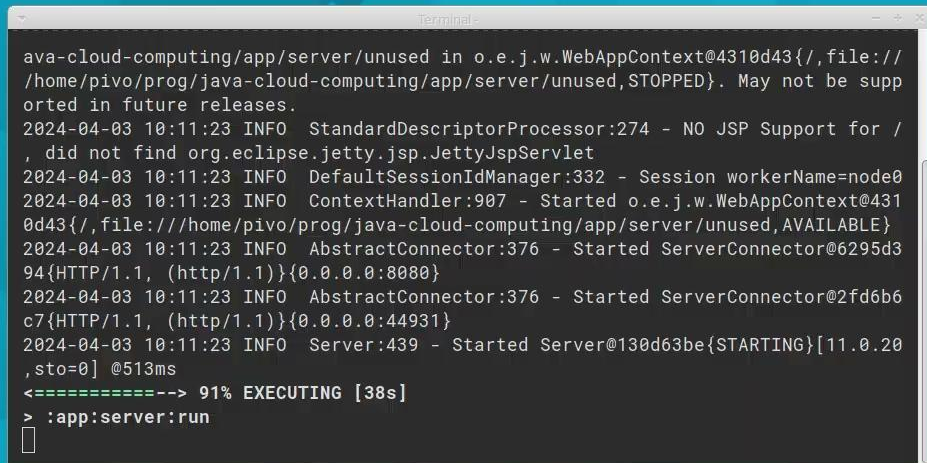
Демо



The screenshot shows an IDE with a dark theme. The top bar includes a menu, a search bar, and a file explorer. The main editor displays the following Java code:

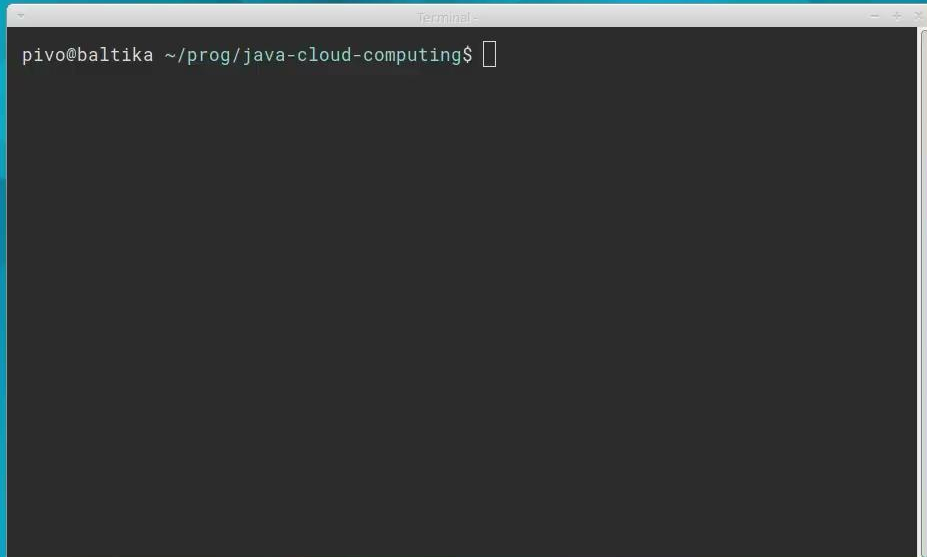
```
1 package ru.nsu.fit.smolyakov.sobakacloud.aop.appexample;
2
3 import ru.nsu.fit.smolyakov.sobakacloud.aop.annotation.SobakaCloudCompute;
4 import ru.nsu.fit.smolyakov.sobakacloud.aop.annotation.SobakaEntryMethod;
5
6 @SobakaCloudCompute(server = "localhost:8080")
7 public class ClassToCalculate {
8     @SobakaEntryMethod
9     public static int calculateMe(String some) {
10         try {
11             Thread.sleep(3000);
12         } catch (InterruptedException e) {
13             throw new RuntimeException(e);
14         }
15         throw new NullPointerException("this is exception");
16     }
17 }
18
```

The bottom status bar shows the file path: `> java > ru > nsu > fit > smolyakov > sobakacloud > aop > appexample > ClassToCalculate`, along with line 18, LF, UTF-8, and 4 spaces.



The terminal window displays the following output:

```
ava-cloud-computing/app/server/unused in o.e.j.w.WebAppContext@4310d43{/,file:///home/pivo/prog/java-cloud-computing/app/server/unused,STOPPED}. May not be supported in future releases.
2024-04-03 10:11:23 INFO StandardDescriptorProcessor:274 - NO JSP Support for /, did not find org.eclipse.jetty.jsp.JettyJspServlet
2024-04-03 10:11:23 INFO DefaultSessionIdManager:332 - Session workerName=node0
2024-04-03 10:11:23 INFO ContextHandler:907 - Started o.e.j.w.WebAppContext@4310d43{/,file:///home/pivo/prog/java-cloud-computing/app/server/unused,AVAILABLE}
2024-04-03 10:11:23 INFO AbstractConnector:376 - Started ServerConnector@6295d394{HTTP/1.1, (http/1.1)}{0.0.0.0:8080}
2024-04-03 10:11:23 INFO AbstractConnector:376 - Started ServerConnector@2fd6b6c7{HTTP/1.1, (http/1.1)}{0.0.0.0:44931}
2024-04-03 10:11:23 INFO Server:439 - Started Server@130d63be{STARTING}[11.0.20,sto=0] @513ms
<=====--> 91% EXECUTING [38s]
> :app:server:run
```



The terminal window shows the command prompt:

```
pivo@baltika ~/prog/java-cloud-computing$
```

А где кластеризация

В качестве временного инфраструктурного решения допустимо использовать nginx:

```
http {  
    upstream semiclustert {  
        ip_hash;  
        server c1.semiclustert.local:8080;  
        server c2.semiclustert.local:8080;  
    }  
  
    server {  
        location / {  
            proxy_pass http://semiclustert;  
        }  
    }  
}
```


Безопасность

Зачем?

Исполнение произвольного кода (RCE) — уязвимость в программной системе, когда злонамеренный пользователь может заставить её исполнять любой машинный код. © В. Педия.

В рамках данной работы задача недопущения уязвимости сводится к отфильтровыванию злонамеренного произвольного кода либо к нейтрализации его злонамеренных действий.

Прочие уязвимости пока что оставим за кадром.

Что?

Приведем искусственный пример вредоносного кода (но без механизмов безопасности даже он справляется с поставленной ему задачей):

```
public static int
absolutelyGuaranteedSafeCalculations(int a, int b) {
    var process = Runtime.getRuntime().exec(
        "/bin/sh -c -- 'rm $0' /home/user/cats.jpg"
    );
    return a + b;
}
// в jdk18 метод Runtime::exec(String) стал deprecated,
// есть много альтернатив.
```

Как?

Предлагается рассмотреть следующие опции:

1. Фильтрация системных вызовов. `seccomp-bpf`
2. Средства самой Java. `SecurityManager`
3. Статический анализ (байт-)кода
4. Безопасность на уровне инфраструктуры. `Docker` и `KO`

seccomp-bpf

seccomp (сокр. от англ. secure computing mode) — один из механизмов безопасности ядра Linux, который обеспечивает возможность ограничивать набор доступных системных вызовов для приложений, а также с помощью механизма cBPF производить сложную фильтрацию вызовов и их аргументов. © *Вики. П.*

Используется в Firefox и elasticsearch.

Поддержка eBPF [пока не очень](#), есть альтернативы, например [Kernel runtime security instrumentation](#).

seccomp-bpf

Для обеспечения безопасности запретим “опасные” системные вызовы для процесса. Однако:

- В GNU/Linux (x86-64) свыше 450 системных вызовов, при этом значительная их часть требует сложной ручной фильтрации по аргументам (они вредоносны только с определенными аргументами).

Пример: безобидное разрешение на вызов `read` позволяет читать `/proc/self/mem` и, соответственно, делать дампы памяти процесса.

seccomp-bpf

- Java-программы запускаются в JVM, которая сама собой использует множество разных системных вызовов, таким образом, легко поломать работу JVM или её предыдущих/будущих версий.

```
pivo@baltika ~$ firejail --noprofile --seccomp.drop=clone3 -- java -version
Seccomp list in: clone3, check list: @default-keep, prelist: clone3,
Parent pid 19936, child pid 19937
Seccomp list in: clone3, check list: @default-keep, prelist: clone3,
Child process initialized in 14.98 ms
[0.005s][warning][os,thread] Failed to start thread "GC Thread#0" - pthread_create failed (EPERM) for attributes:
stacksize: 1024k, guardsize: 4k, detached.
#
# There is insufficient memory for the Java Runtime Environment to continue.
# Cannot create worker GC thread. Out of system resources.
# An error report file with more information is saved as:
# /home/pivo/hs_err_pid4.log

Parent is shutting down, bye...
```

SecurityManager

Механизм Java-машины, предоставляющий средства для определения политик безопасности, включая проверку разрешений на выполнение таких операций, как доступ к файлам, сетевым и другим системным ресурсам.

В целом подход схож с предыдущим, однако фильтрация происходит не на уровне ОС, а внутри самой JVM.

SecurityManager

Искусственная защита от искусственного примера:

```
public class CatsSecurityManager extends SecurityManager {  
    @Override  
    public void checkExec(String cmd) {  
        throw new SecurityException("низзя");  
    }  
}
```

SecurityManager

Однако:

- Deprecated for removal начиная с jdk17 [[JEP 411](#)]
- Имеет проблемы с производительностью (впрочем, как и остальные способы)
- Как и ранее, крайне сложно учесть всевозможные ограничения и не оставить дыру в безопасности

Статический анализ кода

Можем перед выполнением методов попытаться найти нелегальные вызовы других методов или какие-то подозрительные конструкции. Однако:

- большая часть уязвимостей зависит от состояния программы (фазы луны) в рантайме
- привет, обфускация
- Миссия невыполнима (1996), реж. Брайан Де Пальма

Статический анализ кода

```
public static int absolutelyGuaranteedSafeCalculations(int, int);
    0: invokestatic #7
// Method java/lang/Runtime.getRuntime:()Ljava/lang/Runtime;
    3: ldc          #13
// String /bin/sh -c -- 'rm $0' /home/user/cats.jpg
    5: invokevirtual #15
// Method java/lang/Runtime.exec:(Ljava/lang/String;)Ljava/lang/Process;
    8: astore_2
    9: iload_0
   10: iload_1
   11: iadd
   12: ireturn
}
```

Docker и КО

Поступающий на сервер код запускается в контейнере.

- Большой оверхед, особенно на старт вычислений
- Зато хост в безопасности
- Хост общается с контейнером через RPC
 - Можно запускать по контейнеру на каждое приходящее вычисление, тогда получается безопасно-безопасно и медленно
 - Можно запускать все вычисления в одном/н контейнерах по очереди, тогда по идее пользователь может поломать что-то внутри контейнера, это надо узнавать и перезапускать контейнер

Спасибо за
внимание!