Application3实验流程： Load generator 模拟发送用户请求到controlvm，通过haproxy负载均衡将请求发送到其他work vm里, workvm的saasi（web interface）接收到请求后，发送消息到business service，business继续将消息分发到对应的api microservice（cpu memory io）处理请求。 Monitor监控 api microservice 的资源消耗并进行scale操作， Business service检测自己是否timeout并将violation信息发送到 controlvm的 globalDM，globalDM将Business service的信息存到数据库中，当达到scale条件时发送请求到DM， DM对business service进行scale。

具体可以看实验论文

部署：

Control vm： 将saasi-expriment.sql导入mysql数据库(数据库名：saasi 账号：root，密码 MyNewPass4!)

将 globalMonitor.war 部署到tomcat里

配置haproxy负载均衡，映射5000端口到其他vm的5001端口（saasi web interface使用）

Workvm ： 安装docker

将Application3 文件夹上传到vm，

运行./run.sh 将saasi、business service、api microservice部署到docker container中

在vm上运行DM

进入Monitor文件夹，运行sudo docker-compose up -d启动cadvisor； 运行monitor （关闭所有程序时先关掉cadvisor 再关掉其他container）

（DM monitor运行在vm上，其他程序运行在docker container里）

运行load generator dotnet run <application type>, <user number>, <request time>

例如 dotnet 3 60 3: 一共60个用户，分成3个批次发送请求

自动化测试：

使用ansible

存在的问题：

* 对io数据的读取不准确，我们监控了block io的读写速率作为scale的标准，是通过对cadvisor的数据计算得出，但只在io microservice container运行的初期比较准确（和 docker stats命令里的数据相比）
* Container之间通过rabbitmq传递消息和分配请求，考虑改成REST请求并用haproxy进行负载均衡