Load generator：

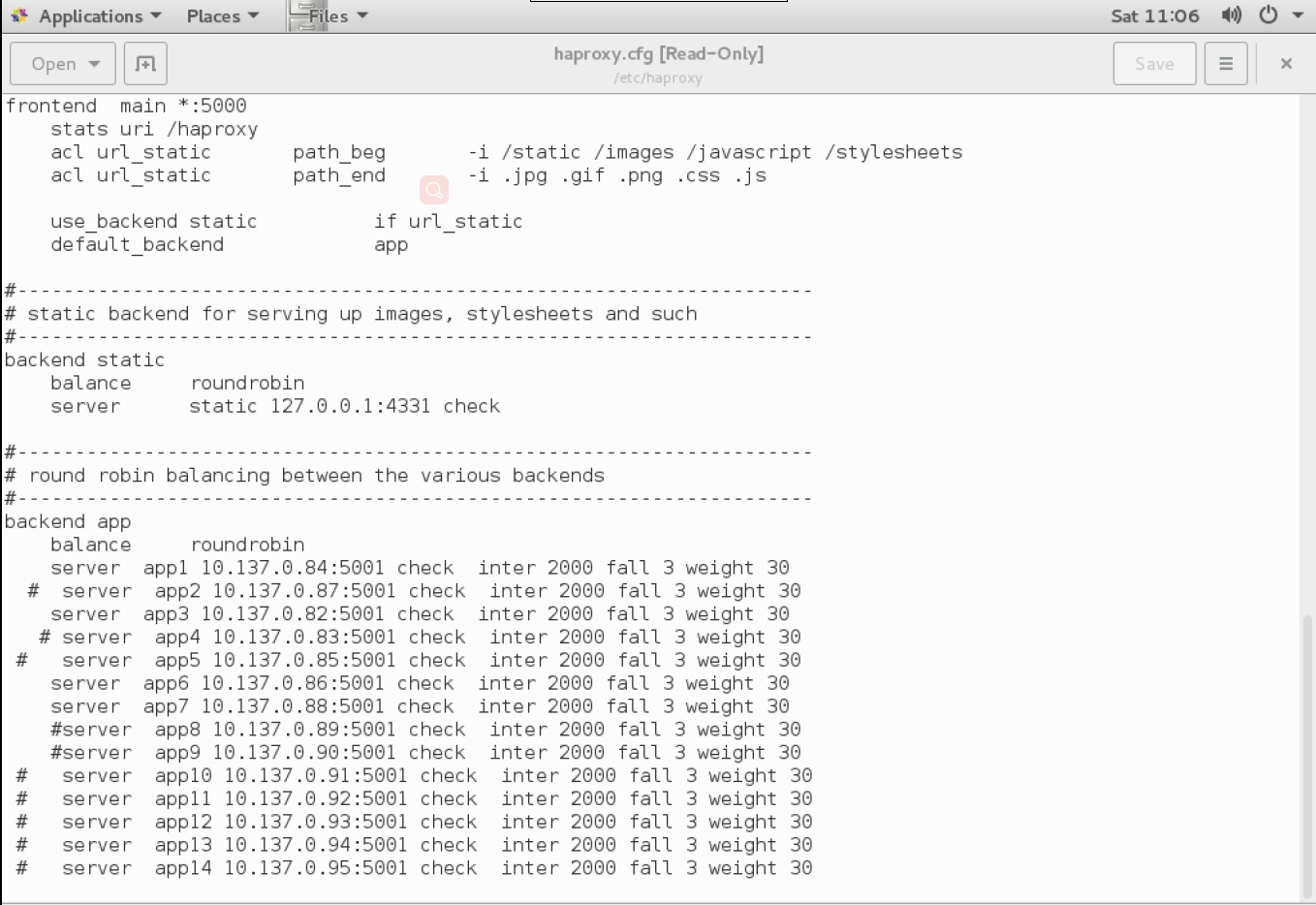
模拟用户发送请求，命令行运行

dotnet run <application type>, <user number>, <request time>

例如 dotnet 3 60 3: 进行application3的实验，模拟60个用户同时发送请求，分成3个批次发送请求

Control VM（当前为saasi-vm01.it.deakin.edu.au）：

* Load balancer (haproxy实现 配置文件 etc/haproxy/haproxy.cfg)



Frontend为haproxy使用的端口，当前为5000，该端口接收到http请求时会转发请求到其他workvm的5001端口。

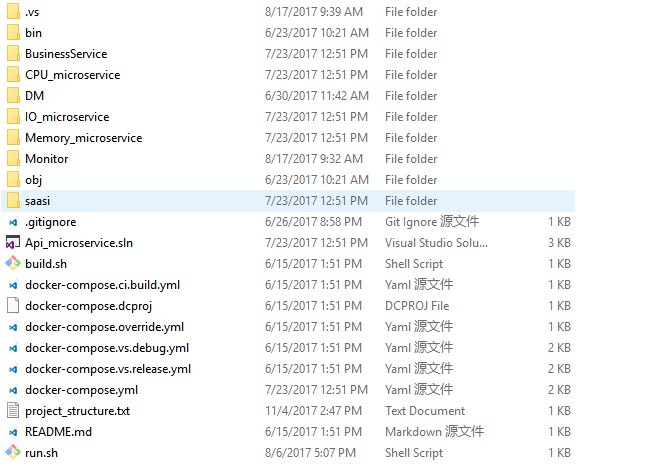
Backend app处控制转发请求的workvm，图示目前能将http请求转发到app1，app3,app6,app7对应的ip端口。

* GlobalMonitor (一个javaweb项目，打包为globalMonitor.war, 放在/usr/local/apache-tomcat-8.5.16下。 目前只用来控制business violation，即当workvm上出现business violation时，会发送请求到globalMonitor，globalMonitor将violation信息记录到数据库，当一个business container的violation次数达到threshold时，globalMonitor发送请求到该business container所在的vm的 Distributed monitor。

实验流程：

1. 部署control vm：导入sql； 将globalmonitor.war放在指定位置，打开tomcat(只用做一次，重复实验时不用再管)

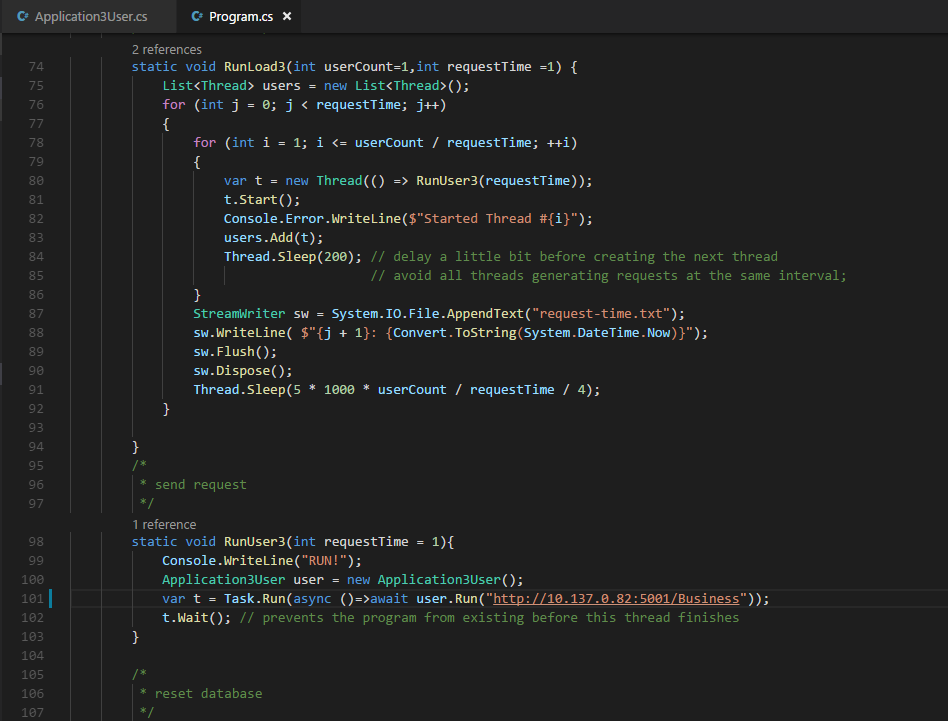
Work VM （application3结构）：



* Business web interface（/saasi, controlvm的请求转发到这，然后通过rabbitmq发送消息到businessservice）
* Business microservice （/BusinessService， 模拟business service，发送消息到api microservice自身检测business violation并发送请求到global monitor）
* API microservice (/cpu,memory,io\_microservice, 模拟对应的intensive操作， 会导致这些container对应的cpu，memory，io使用率上升)
* Distributed monitor interface (/DM, distributed monitor web interface, 接受从global monitor发来的business scaleout请求)
* Distributed monitor (/Monitor, 监控api microservice container的资源消耗情况，若达到threshold则进行scale)

1. 部署work vm：
2. 运行./run.sh, 将Business web interface，business microservice 和 api microservice部署为docker container
3. 进入DM文件夹，命令行运行Distributed monitor interface （重复实验时这个可以不关）
4. 进入Monitor文件夹，运行sudo docker-compose up -d启动cadvisor； 运行monitor （重复实验时先关掉cadvisor， 再关掉其他docker container）
5. 在自己的电脑或者control vm上运行load generator
6. 等到所有用户请求执行完毕后，在DM/DM/data文件夹下会有business violation.txt和business scale.txt; Monitor/data文件夹下会有api-scaleout.txt, cpuStat.txt, ioStat.txt, memStat.txt， 收集这些数据。

在开始测试时可以只用到一台workvm，在load generator里直接修改发送请求的url



如图所示请求直接发送到vm02上

实验过程：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business 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，将数据存入txt文件。 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，将数据存入txt文件。

自动化实验：

在control vm上安装ansible，运行ansible脚本。目前存在的问题时从github pull项目后，linux下dotnet build输出的目录和dockfile输出目录不一致，导致docker container无法正常生成。这个问题解决后可以自动部署和测试整个实验。