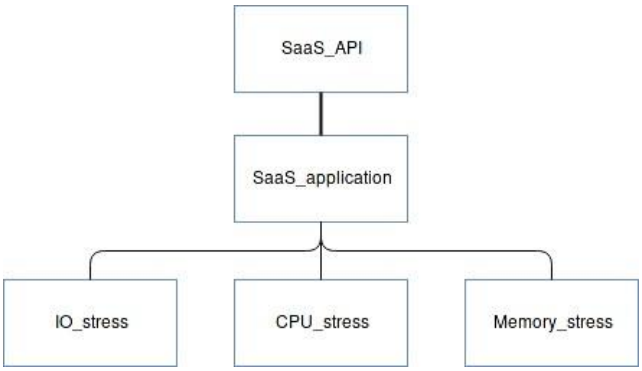
# Artificiële SaaS-applicatie

Er is bewust voor gekozen om geen bestaande SaaS-applicatie te gebruiken om onverwachts complex gedrag, zoals bugs, van dergelijke applicaties te vermijden. Daarom hebben we zelf een artificiële SaaS-applicatie ontwikkeld, die als doel heeft om een mutli-tenant SaaS-applicatie in zijn meest fundamentele vorm te simuleren. In dit hoofdstuk wordt het ontwerp en de implementatie van deze applicatie besproken.

## 4.1 Ontwerp




Figuur 4.1: Ontwerp SaaS-applicatie

Op figuur 4.1 staat het klassediagram. Het ontwerp van de applicatie leunt aan bij de COMITRE aanpak [31]. Er worden in COMITRE 7 stappen gedefinieerd. Stap 2 en 3 worden afgehandeld door onderliggende software. De andere stappen zijn mee in het ontwerp van de SaaS-applicatie verwerkt.



De applicatie is geschreven in C++ en biedt een REST API (`SaaS_API`) waarop tenants requests kunnen sturen (stap 1 in COMITRE). Deze requests bevatten een tenantId opdat de SaaS-applicatie de juiste tenant specifieke configuratie kan terug vinden (stap 4 in COMITRE). Wanneer geen configuratie beschikbaar is, wordt er teruggevallen op een standaard configuratie (zie stap 5). Op basis van de tenant specifieke instellingen worden de stressfuncties met de juiste parameters opgeroepen (zie stap 6).

De applicatie kan geconfigureerd worden om in multi-tenant of single-tenant mode te draaien. In multi-tenant mode is er een cache voorzien, deze wordt gebruikt om elke request tenant specifieke configuratie in te laden. Indien een tenant niet in de cache zit, wordt de configuratie file van de SaaS-applicatie nog eens ingelezen om het ophalen van tenant specifieke data te simuleren. De cache wordt op FIFO wijze gevuld. Wanneer de cache vol zit wordt de tenant die er al het langst in zit er uit gehaald.

Daarnaast kunnen er ook voor elke tenant apart parameters geconfigureerd worden om te bepalen welke resourcetypes (CPU, memory of disk I/O) voornamelijk gestrest zullen worden.

De applicatie kan geconfigureerd worden aan de hand van een YAML file. Een voorbeeld wordt gegeven in codeblock 4.1. De resourceparameters (CPU, memory en I/O) kunnen at run-time ook via de rest API ingesteld worden.

```
multi :    true # Zetapplicatiein multi−tenant mode cache_size : 10 # cache     size
           in multi−tenant mode mem_intensity : 100 # sizeof memory to      allocate
           in          bytes
cpu_intensity : 200 io_intensity : 500 # size of f i l e to read/write in Kbytes (must be more then 10)
tenants : # Individuele                configuratie        voor tenants
      1: # tenant id mem_intensity :
           120 cpu_intensity : 110
           io_intensity : 0
      2:
             mem_intensity :     300
           cpu_intensity : 0 io_intensity : 0
      3:
           mem_intensity : 0
           cpu_intensity : 0
           io_intensity : 300
```

Code 4.1: SaaS-applicatie YAML configuratie voorbeeld

## 4.2 Implementatie

Voor de implementatie voor de verschillende stresstests is onder andere gekeken naar algemene benchmarks als lmbench[29], sysbench en stress-ng[39]. Deze hebben echter het doel om een systeem volledig te belasten en configureerbaarheid was hierbij niet



altijd mogelijk. De implementaties van deze benchmarks kwamen wel overeen met de methodes die Matthews et al. [27] hebben gebruikt om performantie-isolatie van virtuele machines te testen. Onze implementatie is dus grotendeels geïnspireerd door dit werk.

### CPU

```
int Cpu_stress : : simulate () { i f ( stress_size_ != 0) { volatile float
    result = stress_size_ ; for ( int i = 0; i< 100 * stress_size_ ; i++){
    result = Cpu_stress : : fac (30) ;
       } return        result ;
    }
}
int Cpu_stress : : fac ( int n) { i f (n == 1)
    {return 1;}
    else {
       int r = n*Cpu_stress : : fac (n - 1) ; return r ;
    }
}
```

Code 4.2: CPU stress code

Voor de implementatie van de CPU stressmethode is gekozen om een relatief eenvoudige bewerking meerdere keren uit te voeren (faculteit van 30). De parameterwaarde die ingesteld kan worden, zal bepalen hoe vaak dit gedaan wordt in de grootteorde 100. Het volatile keyword is nodig om compileroptimalisaties te vermijden.

### Memory

```
void StressMemory : : run ()
{
     int             iterations = memorySize_ / 10;
    i f               ( iterations == 0) { iterations = 1;}
    int memory_block = (memorySize_ * BYTE) / iterations ; // Iteratively allocate
    memory in blocks of 10B for ( int i = 0; i < iterations ; i++) {
        buffer . push_back( ( void *) malloc (memory_block) ) ;
        //Puts 1s in to the      allocated memory so the      allocated memory marked as in use .      is
        memset( buffer . at ( i ) , 1 , memory_block) ;
    }
} void StressMemory : : release (){ i f (memorySize_ != 0)
{ // Free the allocated memory int iterations =
memorySize_ / 10;
```

*4.*

## *Artificiële SaaS-applicatie*

```
        i f                              ( iterations == 0) { iterations = 1;}   i != buffer . size () ;              i++) {
        for ( std : : vector<void *>:: size_type i = 0; free ( buffer . at
            ( i ) ) ;
        }
    }
}
```

Code 4.3: Memory stress code

Om memory toegang te testen is er gekozen om een bepaald geheugeblok te alloceren en te vullen met één'tjes. Dit wordt gedaan in blokken van 10 Byte om incrementeel inladen van data te simuleren. Er kan ingesteld worden hoeveel byte gebruikt wordt door de applicatie.

### I/O

```
void Io_stress : : run () { i f ( stress_size_ != 0){ std : : ofstream saas_out (              i++){
    "/tmp/saas_out " ) ; for ( int i = 0; i < (KILOBYTE / 4) * stress_size_ ;
                saas_out << "1" <<          "\n" ;
      } saas_out . close () ;
}
```

Code 4.4: I/O stress code

Het uitvoeren van een I/O operatie is hier wegschrijven van data. Dit simuleert het wegschrijven van een resultaat voor een request. De parameter die ingesteld kan worden bepaald hoe groot de file is die weggeschreven wordt in kilobyte.

### API

Requests naar het programma kunnen gestuurd worden op een REST API. Naar /request in single tenant mode, en naar /request/<tenant id> in multi-tenant mode. Voor elke request worden volgende operaties uitgevoerd:

```
    mem_stress->run () ; cpu_stress->run () ;
    io_stress->run () ; mem_stress->release () ;
```

De redenering hierbij is dat een request een bepaalde hoeveelheid RAM-geheugen vereist, een berekening doet, deze wegschrijft en dan het gebruikte geheugen terug vrijgeeft. Indien voor een resource de parameterwaarde op nul is ingesteld, wordt er voor die resource geen code uitgevoerd.

## Detailed API of the multi-tenant SaaS:

The interface in C++ is as follows:

```
#ifndef SaaS_app_H
#define SaaS_app_H
```

```
3  class SaaS_API;
4  #include "SaaS_API.h"
5  #include "yaml-cpp/yaml.h"
6
7  class Saas_application {
8    public:
9
10     Saas_application();
11
12     void print_application_config();
13
14     bool get_multi();
15     int get_mem_intensity(int);
16     int get_io_intensity(int);
17     int get_cpu_intensity(int);
18     int get_cache_size();
19
20     void set_mem_intensity(int, int);
21     void set_io_intensity(int, int);
22     void set_cpu_intensity(int, int);
23     void set_cache_size(int);
24
25     int single_tenant_request();
26     int multi_tenant_request(int);
27
28   private:
29     std::vector<int> cache;
30     YAML::Node config;
31     void simulate(int, int, int);
32     void tenant_lookup(int);
33 };
34
35 #endif
```

The REST API is defined as follows:

```
#include
<string>
```

```cpp
#include <iostream>
#include "SaaS_API.h"
#include "lib/crow_all.h"
#include "Memory_stress.h"

SaaS_API::SaaS_API(Saas_application* application):
  application(application)
{}

// /set_mem/id/int
void SaaS_API::setter_api(crow::SimpleApp& app) {

  CROW_ROUTE(app, "/set_mem/<int>/<int>")
    ([this](int id, int mem_intensity) {
     CROW_LOG_INFO << "Setting mem param: " << mem_intensity;
     application->set_mem_intensity(id, mem_intensity);
     return "mem param has been set";
    });

  CROW_ROUTE(app, "/set_io/<int>/<int>")
    ([this](int id, int io_intensity) {
     CROW_LOG_INFO << "Setting io param: " << io_intensity;
     application->set_io_intensity(id, io_intensity);
     return "io param has been set";
    });

  CROW_ROUTE(app, "/set_cpu/<int>/<int>")
    ([this](int id, int cpu_intensity) {
     CROW_LOG_INFO << "Setting cpu param: " << cpu_intensity;
     application->set_cpu_intensity(id, cpu_intensity);
     return "cpu param has been set";
    });

  CROW_ROUTE(app, "/set_cache/<int>")
    ([this](int cache_size) {
```

```
37      CROW_LOG_INFO << "Setting cache param: " << cache_size;
38      application->set_cache_size(cache_size);
39      return "cache param has been set";
40     });
41 }
42
43 void SaaS_API::multitenant_api(crow::SimpleApp& app) {
44
45   CROW_ROUTE(app, "/request/<int>")
46     ([this](int tenant_id) {
47      // Maybe do not allow requests for id 0, as it is used as a reserved default id
48      CROW_LOG_INFO << "Multitentant request for id: " << tenant_id;
49      application->multi_tenant_request(tenant_id);
50      return "succes";
51     });
52 }
53
54 void SaaS_API::single_api(crow::SimpleApp& app) {
55   CROW_ROUTE(app, "/request/")
56     ([this]() {
57      CROW_LOG_INFO << "Single tenant request";
58      application->single_tenant_request();
59      return "succes \n";
60     });
61 }
62
63 void SaaS_API::expose() {
64   crow::SimpleApp app;
65   crow::logger::setLogLevel(crow::LogLevel::CRITICAL);
66   setter_api(app);
67   if (application->get_multi()) {
68     multitenant_api(app);
69   } else {
70     single_api(app);
71   }
```

```
    app.port(5000).multithreaded().run();
}
```

All requests are GET requests

To send a message to the API you have to use a GET request, or simply type in your browser, for example the following URL, http://<kubeserviceip>/setmem/0/25. In linux the following command does the job `wget http://<kubeserviceip>/setmem/0/25`

/setmem/0/250  means you run the application as a single tenant and you set the time intensity of the algorithm for the entire application to 25

`wget http://<kubeserviceip>/request` is basically the invocation of a single request

Using pyton,  such request can be sent using the package urllib as follows:

```
self.path = "/request/" + str(tenant.tenant_id)
self.request_url = "http://" + self.ip + ":" + self.port + self.path

try:
            urllib.request.urlopen(self.request_url).read()
except:
            self.timeout_count += 1
            print("timeout + 1: " + str(self.timeout_count))
```

The example-controller in github already implements a Request class that uses a Java REST client to invoke the API.

The semantics of the API is best described by reading the thesis. For full reference, here's the implementation of the API

It distinguishes between running the application as a multi-tenant application or as a single tenant application.

```
Saas_application::Saas_application()
{
  config = YAML::LoadFile("saas_config.yaml");

  SaaS_API* api = new SaaS_API(this);

  api->expose();
}

int Saas_application::single_tenant_request() {
  simulate(get_mem_intensity(0), get_cpu_intensity(0), get_io_intensity(0));
}
```

```
24
25  int Saas_application::multi_tenant_request(int tenant_id) {
26    tenant_lookup(tenant_id);
27    // TODO: getters op basis van id maken.
28    simulate(get_mem_intensity(tenant_id), get_cpu_intensity(tenant_id),
      get_io_intensity(tenant_id));
29  }
30
31  void Saas_application::tenant_lookup(int tenant_id) {
32
33    if (std::find(cache.begin(), cache.end(), tenant_id) == cache.end()) {
34      //perform I/O
35      std::ifstream in("saas_config.yaml");
36      if (in.is_open())
37      {
38        CROW_LOG_INFO << "Id not in cache";
39        std::string line;
40        while ( getline(in,line) )
41        {
42          if (line.find("multi") == 0 ){
43            CROW_LOG_INFO << "Performing I/O";
44          }
45        }
46        in.close();
47      }
48
49    // Add to cache
50      if(cache.size() >= get_cache_size()) {
51        cache.erase(cache.begin());
52      }
53      cache.push_back(tenant_id);
54      std::cout << cache.size() << std::endl;
55    }
56  }
57
58  void Saas_application::simulate(int mem, int cpu, int io) {
```

```
59    StressMemory* mem_stress = new StressMemory(mem);
60    Cpu_stress* cpu_stress = new Cpu_stress(cpu);
61    Io_stress* io_stress = new Io_stress(io);
62
63    mem_stress->run();
64    cpu_stress->run();
65    io_stress->run();
66    mem_stress->release();
67    delete mem_stress;
68    delete cpu_stress;
69    delete io_stress;
70  }
71
72  void Saas_application::print_application_config() {
73
74    std::cout << "multitenancy: "<<  get_multi() << std::endl;
75    std::cout << "mem_intensity: " <<  get_mem_intensity(0) << std::endl;
76    std::cout << "cpu_intensity: " <<  get_cpu_intensity(0) << std::endl;
77    std::cout << "io_intensity: "  <<  get_io_intensity(0) << std::endl;
78
79  }
80
81  bool Saas_application::get_multi() {
82    return config["multi"].as<bool>();
83  }
84
85  int Saas_application::get_mem_intensity(int id) {
86    if (config["tenants"][id]) {
87      return config["tenants"][id]["mem_intensity"].as<int>();
88    } else {
89      return config["mem_intensity"].as<int>();
90    }
91  }
92
93  int Saas_application::get_io_intensity(int id) {
94    if (config["tenants"][id]) {
```

```
    return config["tenants"][id]["io_intensity"].as<int>();
  } else {
    return config["io_intensity"].as<int>();
  }
}

int Saas_application::get_cpu_intensity(int id) {
  if (config["tenants"][id]) {
    return config["tenants"][id]["cpu_intensity"].as<int>();
  } else {
    return config["cpu_intensity"].as<int>();
  }
}

int Saas_application::get_cache_size() {
  return config["cache_size"].as<int>();
}


void Saas_application::set_mem_intensity(int id, int mem_int) {
  CROW_LOG_INFO << "setting mem_intensity " << mem_int << " for id: " << id;
  if (config["tenants"][id]) {
    CROW_LOG_DEBUG << "setting for id: " << id;
    config["tenants"][id]["mem_intensity"] = mem_int;
  } else if (id == 0) {
    CROW_LOG_DEBUG << "setting default";
    config["mem_intensity"] = mem_int;
  } else {
    CROW_LOG_WARNING << "Not a valid id: " << id << ". To change the default value, use id 0";
  }
}

void Saas_application::set_io_intensity(int id, int io_int) {
  CROW_LOG_INFO << "setting io_intensity " << io_int << " for id: " << id;
  if (config["tenants"][id]) {
    config["tenants"][id]["io_intensity"] = io_int;
```

```
  } else if (id == 0) {
    config["io_intensity"] = io_int;
  } else {
    CROW_LOG_WARNING << "Not a valid id: " << id << ". To change the default value, use id 0";
  }
}

void Saas_application::set_cpu_intensity(int id, int cpu_int) {
  CROW_LOG_INFO << "setting cpu_intensity " << cpu_int << " for id: " << id;
  if (config["tenants"][id]) {
    config["tenants"][id]["cpu_intensity"] = cpu_int;
  } else if (id == 0) {
    config["cpu_intensity"] = cpu_int;
  } else {
    CROW_LOG_WARNING << "Not a valid id: " << id << ". To change the default value, use id 0";
  }
}

void Saas_application::set_cache_size(int cache_size) {
  config["cache_size"] = cache_size;
}
```