

Software Defined Communication Infrastructure

Distributed System and Big Data - INSA Toulouse - 2023 **Théo Fontana** , **Jose Organista**

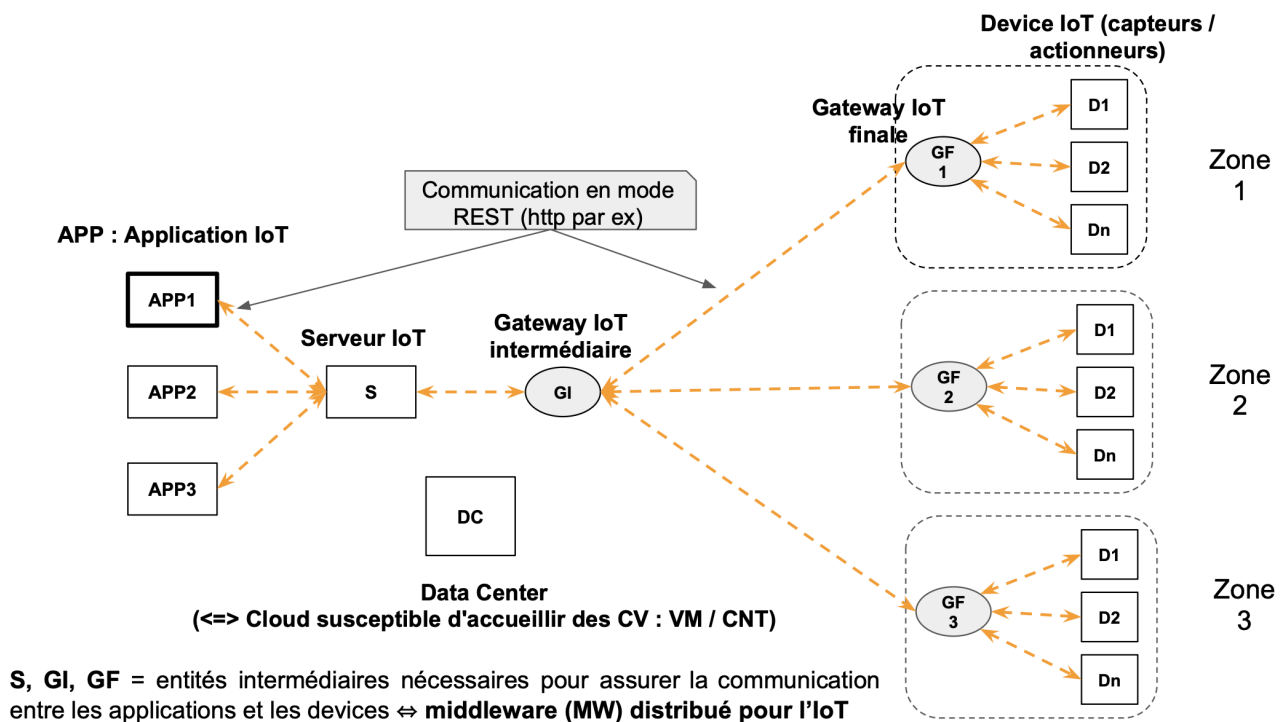
Presentation du Projet

Objectifs

- Déployer dynamiquement et de façon transparente des fonctions de réseau virtuelles (VNF)
 - permettant de répondre aux besoins fonctionnels et/ou non fonctionnels d'applications distribuées relevant d'une activité de l'Internet des objets (IoT)
 - en appliquant les concepts et techniques relevant de la virtualisation de fonctions de réseau (NFV) et des réseaux pilotables par le logiciel (SDN)
- Développer une approche de gestion autonome de la mise en œuvre des VNF ciblées via le concept de l'Autonomic Computing (AC)

Activité IoT ciblée

Activité de supervision/intervention à distance sur différentes zones dotées de capteurs / actionneurs, par le biais d'applications



En cas d'incident dans une zone du trafic supplémentaire est généré par ses capteurs / actionneurs. Ceci peut entraîner la saturation de la gateway intermédiaire (GI) générant ainsi une baisse de performances incompatible avec les besoins en QoS des applications.

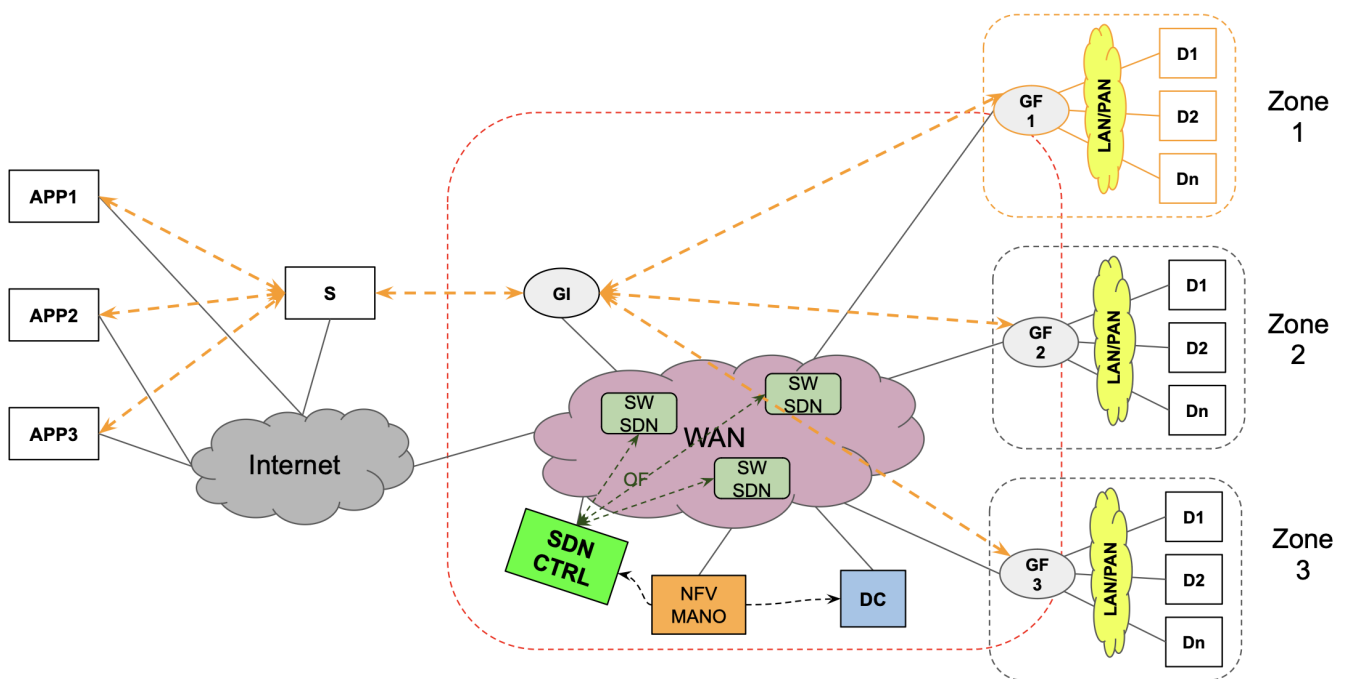
Une phase d'adaptation est alors nécessaire pour rétablir les performances. Plusieurs stratégies peuvent être adoptées :

- Déployer une seconde gateway sous forme de VNF et rediriger le trafic provenant de la zone 1 (ou des zones 2 et 3) vers cette gateway.
- Déployer d'une VNF d'ordonnancement différencié priorisant le trafic issu de GF1.
- Supprimer les flux de données en provenance de la zone 2 et 3.
- Déployer d'un loadbalancer sous forme de VNF.

Vision IT de l'activité ciblée

Hypothèse sur l'infrastructure IT

- GI, GF et DC sont connectés via un réseau grande distance (WAN) géré par un opérateur dont la portée d'action inclut : les nœuds internes du réseau (switch), les nœuds MW (GI et GF) et le DC
- Un orchestrateur de VNF (VNF-ORCH) est connecté au WAN : il permet de déployer des VNF sur le DC et de gérer leur cycle de vie.
- Le WAN est doté de capacités SDN :
 - Ses nœuds internes sont des switch SDN programmables via Open Flow
 - Il inclut un contrôleur SDN interagissant avec les switch SDN via Open Flow



Plateforme et outils mis à disposition

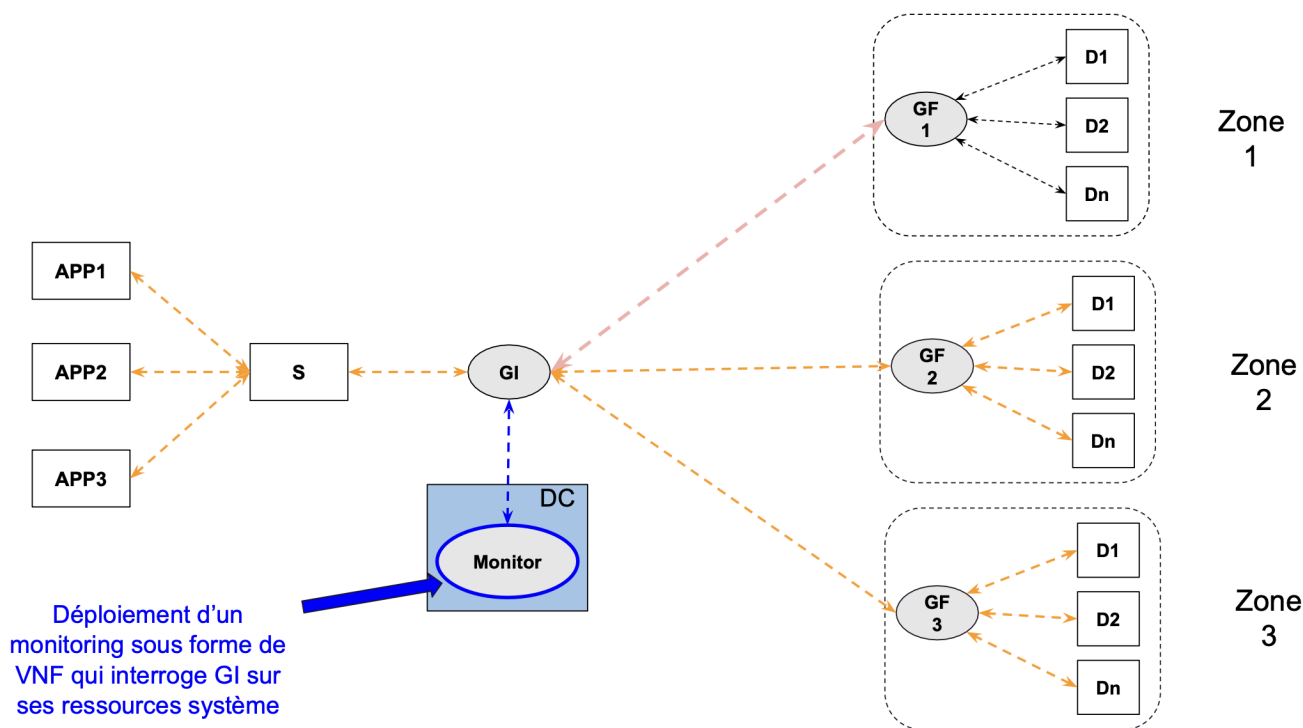
- Plateforme d'émulation de réseau : [ContainerNet](#)
- Contrôleur SDN : [RYU](#)
 - [documentation](#)
- MANO standardisé ETSI NFV : [OSM](#)
- Middleware IoT/M2M en NodeJS (see [Middelware](#))

Travail demandé

Mettre en place l'adaptation requise lorsque la gateway intermediaire est saturée, suivant le cadre de l'Autonomic Computing

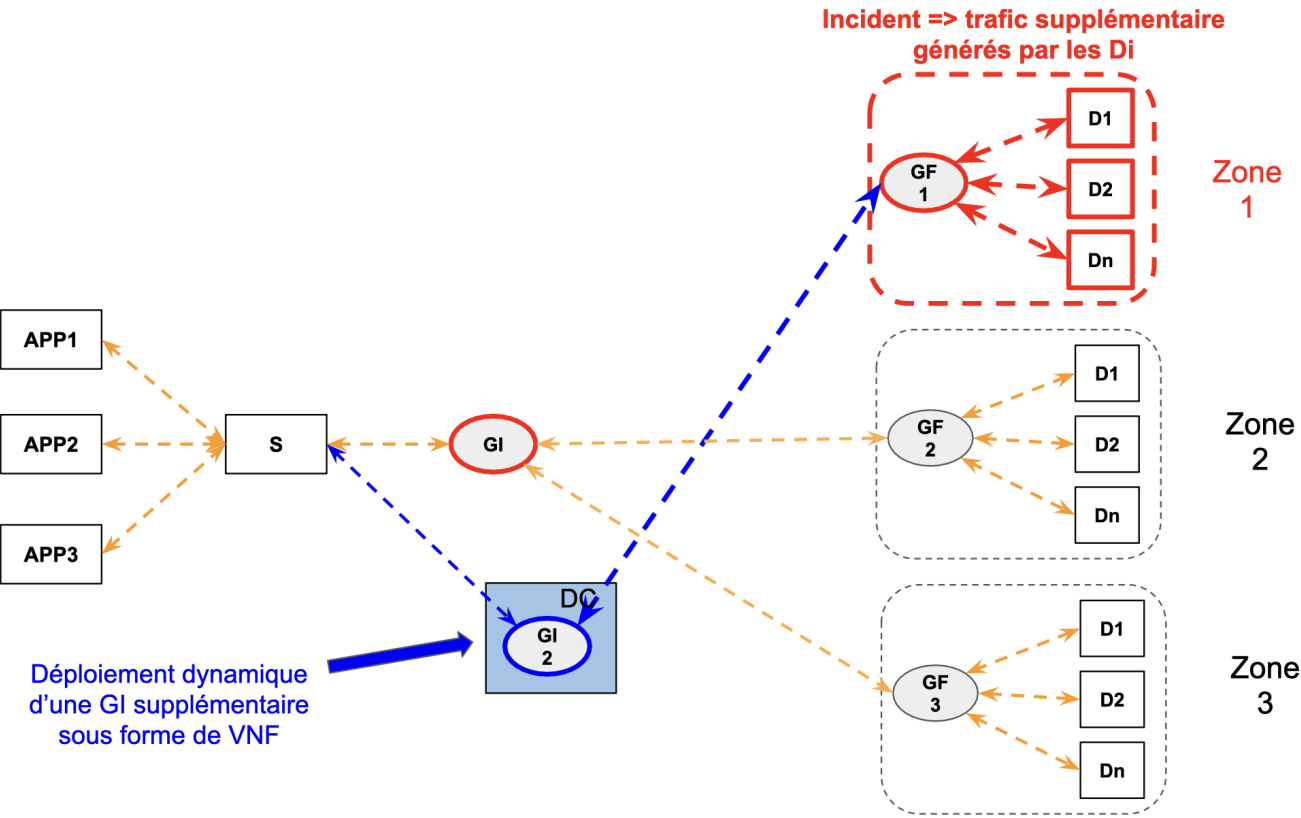
Use cases étudiée

Notre groupe avait pour mission de monitorer la gateway intermediaire pour surveiller sa charge à partir de metrique système tel que la charge du CPU.



Nous devons en suite en cas de dégradation des performance déployer une nouvelle gateway et redireiger le trafic en provenance de la zone 1 vers cette dernière. Le trafic de la zone 2 et 3 continue d'utiliser la

gateway initiale.



Conception des solutions

Composants en jeu

Nous disposons d'un Mano qui nous expose un service permettant de déployer et d'arreter des VNF dans un datacenter via des requettes sur son API REST.

Nous avons egalement un controller SDN qui nous permet de mettre à jour les tables SDN des differents switch de notre reéseau via son API REST.

Les iterations entre notre general controller, le MANO et le SND controller sont resumé dans le diagramme de structure composite suivant

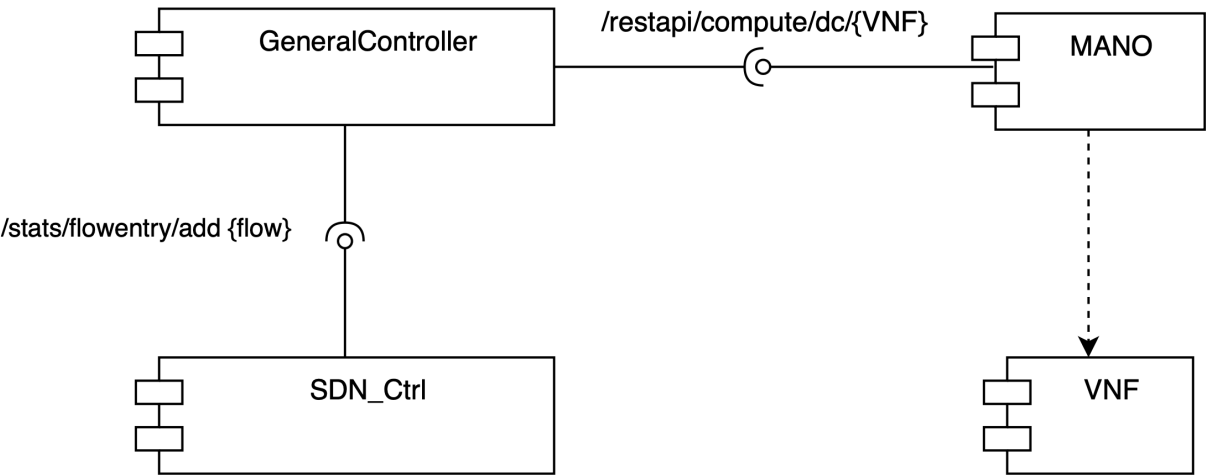


Diagramme de structure composite

Monitoring

Pour le monitoring nous proposons de déployer la VNF de monitoring au démarrage du general controller. Une fois que celui ci à eu la conformation que le VNF est correctement déployé, il l'interrooge periodiquement pour recupérer les informations système de la gateway intermediaire. Il verifie à chaque iteration que le système n'est pas en surcharge.

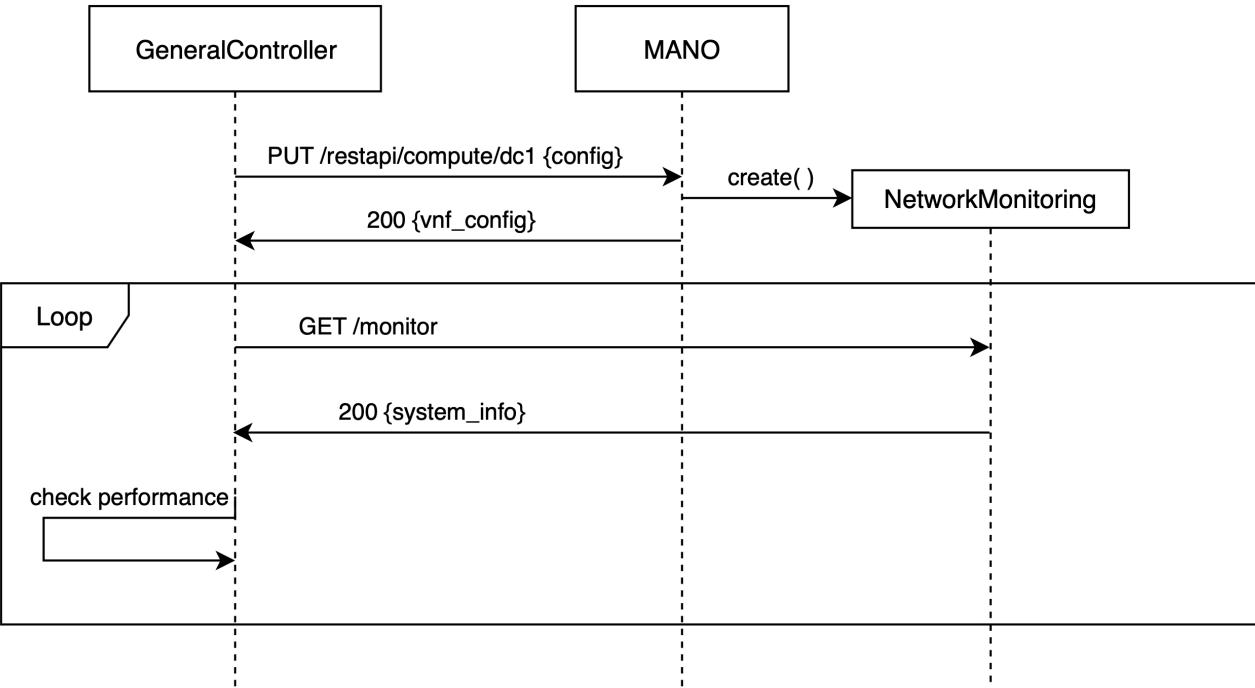


Diagramme de séquence du monitoring

Adaptation

Pour l'adaptation, notre general controller devra demander le deploiment d'une nouvelle gateway dans le datacenter via l'API du MANO. Si ce deploiment s'st bien déroulé, il demande la redirection du trafic de la zone 1 en direction de cette VNF grace à l'API du controller SDN.

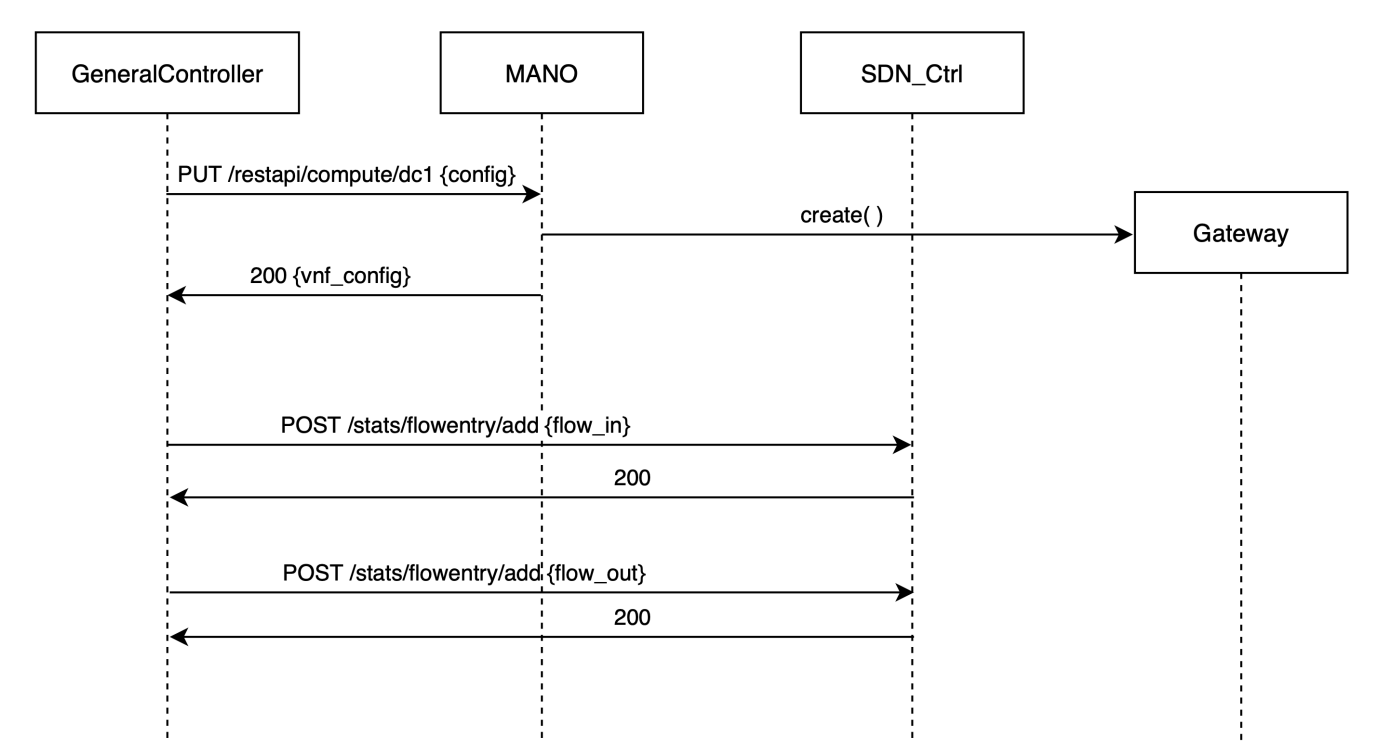
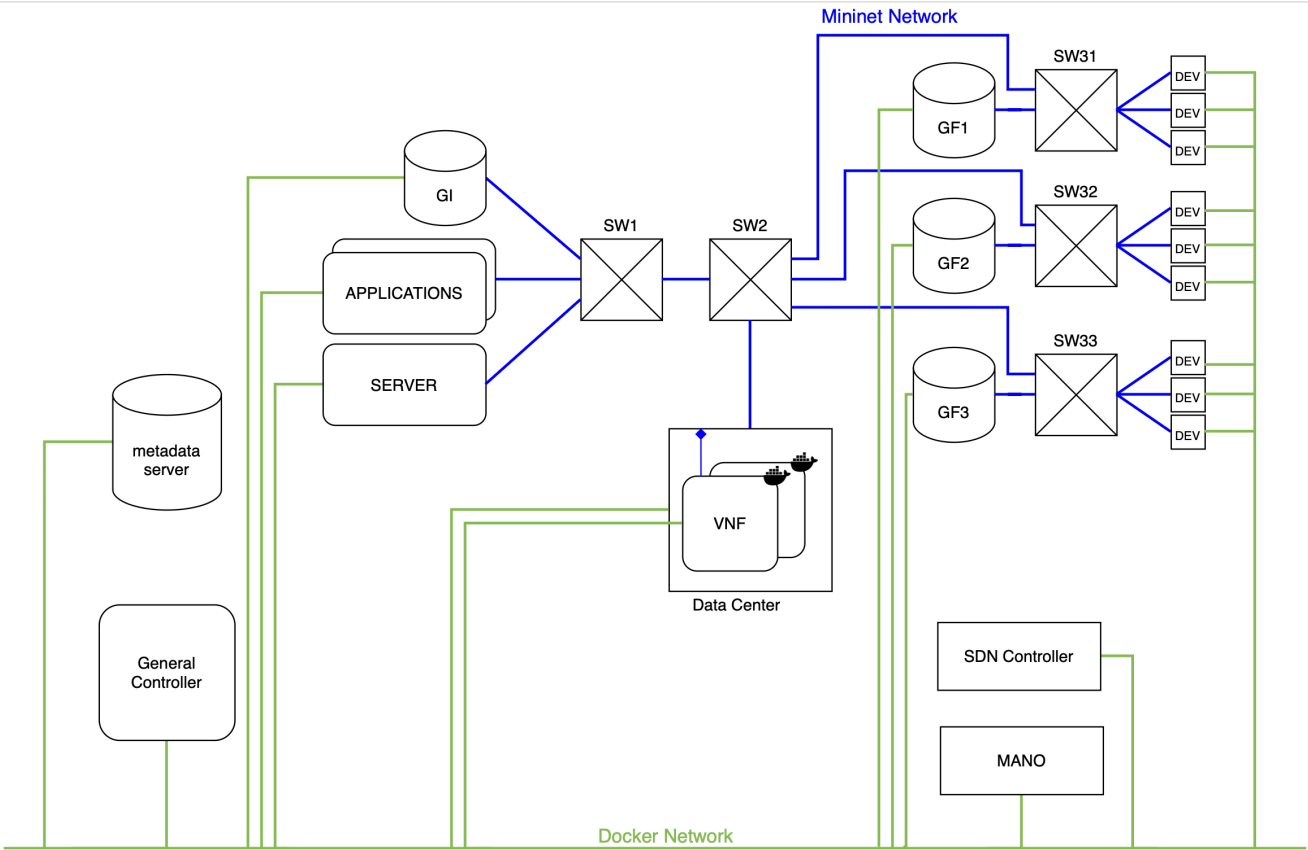


Diagramme de séquence de l'adpatation

Choix d'implementation

Topologie deployé

Nous avons choisi de déployer le reseau suivant



Le reseau bleu est le réseau émulé mininet. Nous avons choisis de simuler les différentes zones avec un switch simulant un LAN.

Le reseau vert représente le réseaux VLAN Docker reliant tous nos containers. Il est utilisé pour assurer la communication entre :

- les noeuds middleware et le metadata serveur
- le GC, le Mano, le controlleur SDN et les VNFs

Pour le deploiment des different noeuds middleware, nous avons créé un unique Dockerfile permettant de créer l'image associée au noeud. Celui ci recupère d'identifiant de l'instance à déployer en variable d'environnement et lance un script de démarrage spécifique en fonction du type d'instance lorsque le container est lancé.

```
FROM ubuntu:trusty

ARG SCRIPT
ARG NODE_VERSION=14
ENV INSTANCE_ID=''
...
ADD $SCRIPT .
...
ENTRYPOINT sh /componnent/$SCRIPT && /bin/sh
```

Extrait du [Dockerfile](#)

Le scrpit de démarage est chargé de récupérer la configuration de l'instance pour pouvoir lancer le service avec les bons paramètres.

```
curl -o conf.json metadata_server/$INSTANCE_ID

LOCAL_NAME=`cat conf.json | jq '.local_name'`
LOCAL_PORT=`cat conf.json | jq -r '.local_port'`
LOCAL_IP=`cat conf.json | jq '.local_ip'`
FILE_URL=`cat conf.json | jq -r '.file_URL'`

curl -LO $FILE_URL
node server.js --local_ip $LOCAL_IP --local_port $LOCAL_PORT --local_name $LOCAL_NAME
```

Exemple de script de demarage pour le serveur [start_server.sh](#)

Metadata serveur

Nous avons réalisé le metadata serveur en Node.js. Il renvoie la configuartion de l'instance à deployer suite à une requette **GET** sur l'identifiant de l'instance souhaité.

```
app.get('/:id', function(req, res) {
  var id = req.params.id;
  var conf_instance = config[id];
```

```

    if (conf_instance)
        res.status(E_OK).send(JSON.stringify(conf_instance));
    else
        res.sendStatus(E_NOT_FOUND);
});

```

Extrait de [metadata_server.js](#)

L'ensemble des configurations est stocké dans un fichier geneneral de configuration json.

```

{
  ...
  "gw1": {
    "local_ip": "10.1.0.11",
    "local_port": 8282,
    "local_name": "gw1",
    "remote_ip": "10.1.0.10",
    "remote_port": 8181,
    "remote_name": "gwi",
    "file_URL": "https://homepages.laas.fr/smedjiah/tmp/mw/gateway.js"
  },
  ...
}

```

Extrait de [config.json](#)

General controleur

Nous avons choisi de ne pas utiliser le squelette de general controller fourni mais de développer un prototype plus simple *from scratch* en Python afin de nous faciliter le developpement et les tests.

Monitoring

Notre strategie de monitoring est pour l'instant assez simple. Lorsque notre VNF recoit une requette **GET** de la part du general controller elle interroge la gateway sur son endpoint **/health** et retourne la reponse reçu au GC. Cette strategie nous permet de déplacer le traitement de la réponse au niveau du GC celui si peut donc choisir à quel rythme monitorer ce qui peut potentiellement reduire la charge sur la gateway.

```

app.get('/monitor', function(req, res) {
    request({method: 'GET', uri: `http://10.1.0.10:8181/health`}, (error,
    response, body) => {
        if (!error && response.statusCode == 200){
            res.send(body);
        } else {
            res.send(error);
        }
    });
});

```


Extrait de *monitor.js*

Pour deployer la VNF, nous utilisons l'API REST de vim-emu

```
def start_monitoring():
    # URL to add new vnf
    url = 'http://127.0.0.1:5001/restapi/compute/dc1/vnf_monitor'
    headers = {'Content-type': 'application/json'}
    d = {"image": "vnf_monitor:0.2", "network": "
(id=vnf_monitor,ip=10.1.0.100/24)"}
    r = requests.put(url, headers = headers, data = json.dumps(d))
    return r.status_code, r.json()
```

Extrait de *controller.py*

Nous avons chois de basé notre monitoring sur la metrique *currentLoadSystem* car c'est celle qui semblait varier le plus rapidement losque nous simulions une charge sur la gateway durant nos tests. Lorque celle ci depasse le seuil fixé, nous devons deployer notre VNF d'adpatation.

Adaptation

La première étape de l'adaption est de déployer une nouvelle gateway intermediaire dans le datacenter en utilisant l'API REST de vim-emu. l'image de la gateway intermediaire precedement construite à du être légèrement modifié pour qu'elle concovienne aux requirements de vim-emu, le serveur node.js doit tourner de background et et les sripts de démarrage et d'arrêt de la VNF doivent être passé en variable d'environnement dans le Dockerfile.

Nous devons ensuite rediriger le trafic de la gateway final de la zone 1 à direction de la gateway intermediaire vers notre VNF.

Nous avons pris la décision d'identider ce flux avec uniquement les adresse IP source et destinations. En effet le seul trafic circulant sur notre réseaux entre ces instance est le trafic applicatif que nous souhaions rediriger. Si ce n'était pas le cas nous aurions également du utiliser les numéros de port pour identifier ces flux.

Nous devons donc :

- modifier l'adresse IP destination des paquet provennat de *GWF_1* en direction de *GWI* (*aller*)
- modifier d'adresse IP srouce des paquets provenant de la *GWI_VNF* en direction en direction de *GWF_1* (*retour*)

Cela est réalisé en ajoutant des *flow* dans la table SDN du switch 2 à l'aide de l'API de controller SDN de la façon suivante (pour l'aller)

```
curl -X POST -d '{
  "dpid": 2,
  "table_id":0,
  "priority":11111,
```

```
"match":{
  "nw_src": "10.1.0.11",
  "nw_dst": "10.1.0.10",
  "dl_type": "2048",

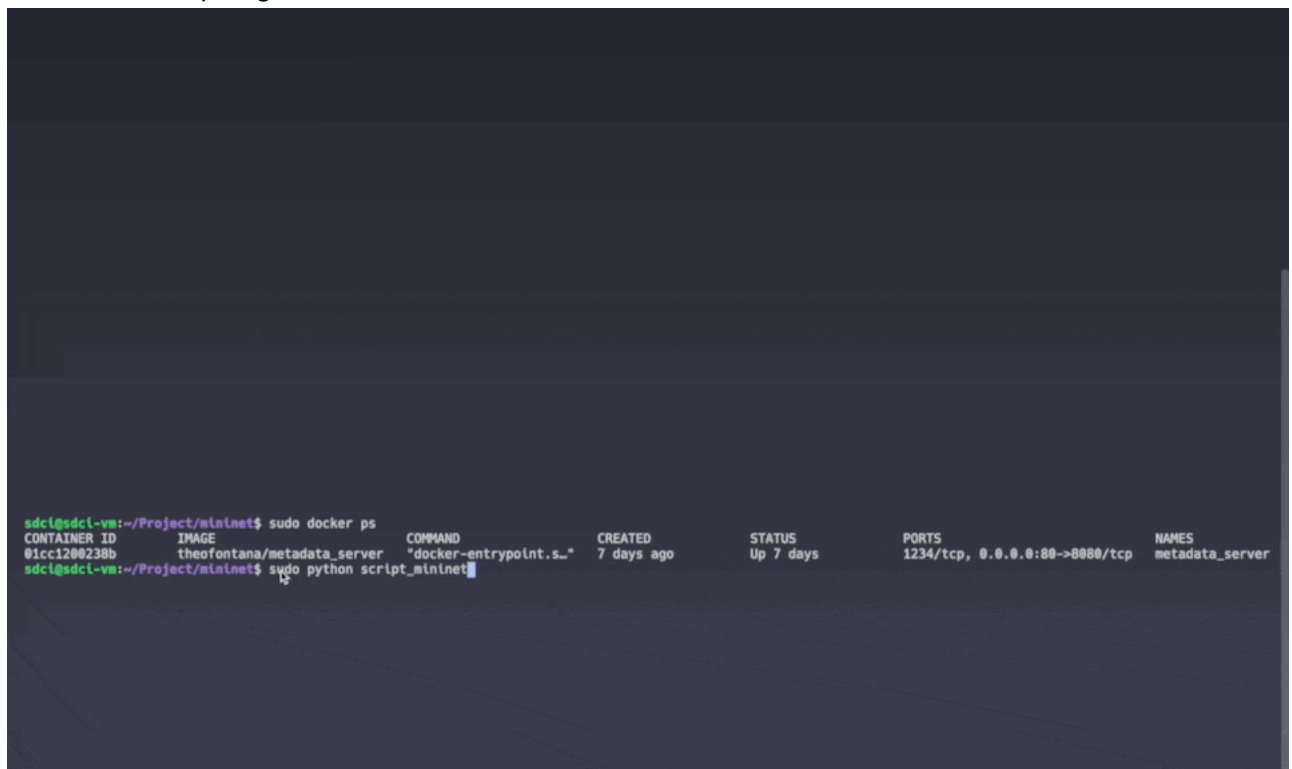
},
"actions":[
  {
    "type": "SET_FIELD",
    "field": "ipv4_dst",
    "value": "10.1.0.60"
  },
  {
    "type": "OUTPUT",
    "port": "NORMAL"
  }
]
}' http://localhost:8080/stats/flowentry/add
```


Extrait de [redirect_gwi_to_vnf.sh](#)

S  nario de d  monstration

Pour notre d  monstartion nous souhaitons

1. Lancer notre topologie avec mininet



- 
- The image displays two side-by-side terminal windows from a virtual machine. Both windows show the command prompt 'sdcl@sdcl-vm:~/Project\$' and the execution of 'python3 main.py'. The left window shows the command being entered, while the right window shows the command being executed, with a cursor visible at the end of the line.

- [illegible]

- 11 / 12

7. Vérifier que ces opérations ont été transparentes au niveau applicatif.

```

sdcl@sdcl-va:~/Project/GC$ python controller.py
({'docker_network': 'u'172.17.0.17'})
({'avgLoad': 0.07})
({'currentLoadSystem': 0.43442279002794837})

DEBUG:dcemulator.net:addLink: n1=s31 intf1=s31-eth6 -- n2=gwf_2 intf2=gwf_2-eth0
DEBUG:dcemulator.net:addLink: n1=s32 intf1=s32-eth2 -- n2=dev_4 intf2=dev_4-eth0
DEBUG:dcemulator.net:addLink: n1=s32 intf1=s32-eth3 -- n2=dev_5 intf2=dev_5-eth0
DEBUG:dcemulator.net:addLink: n1=s31 intf1=s31-eth7 -- n2=gwf_3 intf2=gwf_3-eth0
DEBUG:dcemulator.net:addLink: n1=s32 intf1=s32-eth5 -- n2=dev_7 intf2=dev_7-eth0
DEBUG:dcemulator.net:addLink: n1=s32 intf1=s32-eth6 -- n2=dev_8 intf2=dev_8-eth0
DEBUG:dcemulator.net:addLink: n1=s32 intf1=s32-eth7 -- n2=dev_9 intf2=dev_9-eth0
*** Configuring hosts
root srv gwf gwf_1 gwf_2 gwf_3 dev_1 dev_2 dev_3 dev_4 dev_5 dev_6 dev_7 dev_8 dev_9
*** Starting controller
c0
*** Starting 7 switches
dc1.s1 fs1 s1 s2 s31 s32 s33 ...
*** Starting CLI:
containernet> DEBUG:dcemulator.node:Starting compute instance u'vnf_monitor' in d
ata center 'dc1'
vnf_monitor: kwargs {'environment': {'VNF_NAME': u'vnf_monitor'}, 'datacenter': d
c1, 'flavor_name': 'tiny', 'ip': '10.0.0.29/0'}
vnf_monitor: update resources {'cpu_quota': -1}
DEBUG:dcemulator.net:addLink: n1=vnf_monitor intf1=vnf-monitor -- n2=dc1.s1 intf2
=dc1.s1-eth2
*** vnf_monitor : ( './start_vnf.sh', )

loadIrq: 0,
rawLoad: 10916330,
rawLoadUser: 7953910,
rawLoadSystem: 2919260,
rawLoadNice: 43160,
rawLoadIdle: 662707320,
rawLoadIrq: 0
},
{
  load: 1.6256072757193971,
  loadUser: 1.184386933223733,
  loadSystem: 0.43547839632814567,
  loadNice: 0.005741946167518742,
  loadIdle: 98.3743927242886,
  loadIrq: 0,
  rawLoad: 10956390,
  rawLoadUser: 7982620,
  rawLoadSystem: 2935070,
  rawLoadNice: 38700,
  rawLoadIdle: 663031120,
  rawLoadIrq: 0
}
}
}

root@gwf_1:/component# curl 10.1.0.10:8181/ping
{"pong":1675424761582}root@gwf_1:/component# curl 10.1.0.10:8181/ping
{"pong":1675424763030}root@gwf_1:/component# curl 10.1.0.10:8181/ping
{"pong":1675424763937}root@gwf_1:/component# curl 10.1.0.10:8181/ping
{"pong":1675424764967}root@gwf_1:/component# curl 10.1.0.10:8181/ping

```

demo pour les points 4, 5, 6 et 7

Axes d'ameliorations

- Actuellement lors de la redirection du trafic le trafic de *retour* entre la VNF gateway intermediaire et la gateway finale est adressé à la gateway intermediaire au niveau MAC. Nous n'avons pas pu debugger ce problème qui fait que la GWI reste saturé même après la redirection de trafic effectué.
- Il nous faudrait ensuite ajouter une strategie pour revenir au cas nominal en supprimant la VNF deployé une fois que le trafic redevient normal.
- Il pourrait également être interessant de monitorer la VNF deployer pour s'assurer que celle ci ne soit pas non plus en surcharge et possiblement deployer une nouvelle gateway intermediaire avec un load balancer en cas de problème.

Conclusion

Ce projet a été l'occasion de nous familiariser avec les concepts de l'autonomus computing dans un contexte IoT où les applications ont des besoin en QoS et génèrent un trafic varibale.

Nous avons pu développer et déployer dynamiquement des VNFs en charge de surveiller l'état d'instances sur le reseau et d'assurer des performances sufisante aux applications.

Grace à SDN nous avons pu dynamiquement modifier le routage au sein de notre réseaux de manière transparente pour les applications.