Samuel Andreo Vilaplana

Mateo Bernal Montoya

Ana Isabel Bueno Gimeno

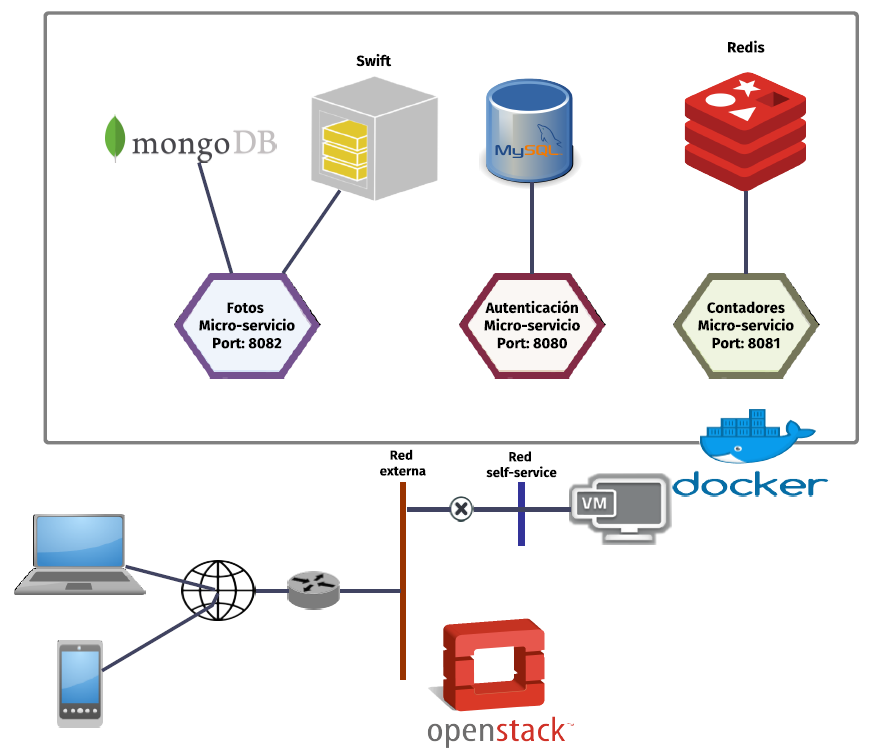
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA - UNIVERSIDAD DE VALENCIA

Computación en la nube

Taller 3: DOcKER swarm

# OBJETIVO DE LA PRÁCTICA.

* Realizar una aplicación con una arquitectura basada en micro-servicios.
* Encapsular los micro-servicios en contenedores.
* Abastecer una máquina virtual en una infraestructura de computación en la nube.
* Definir los servicios que componen la aplicación en un fichero YAML.
* Realizar el despliegue de los servicios en la máquina virtual.
* La arquitectura se muestra en la siguiente figura:



# Desarrollo de los microservicios.

## sb-login (Java - Spring Boot) 8080. Servicio de autenticación.

SpringBoot microservicio para recuperar un token válido. Endpoints:

* POST /login
* POST /register

Modelo: {long id String user String password String rol}

Hay que incluir el siguiente header con el token recuperado en el resto de peticiones:

*Authorization 'Bearer :tokenGivenFromLogin';*

Se conecta al microservicio Mysql

* Build image: docker build -t {nombre\_imagen} .
* Run Container para desarrollo: docker run --name {nombre\_contenedor} -p 8080:8080 -e SECRET\_TOKEN=secretlogin {nombre\_imagen}

|  |
| --- |
| **Dockerfile** |
| #jdk extra ligero oficial  Hacemos uso de un script de espera para que la aplicación se ejecute cuando la conexión a la bbdd (mysql) esté lista para aceptar peticiones. (Créditos a Eduardo Fonte)  FROM openjdk:8-jdk-alpine  VOLUME /tmp  EXPOSE 8080  ADD /target/login-v1.jar app.jar  ## Add the wait script to the image  ADD https://github.com/ufoscout/docker-compose-wait/releases/download/2.2.1/wait /wait  RUN chmod +x /wait  CMD /wait && java -jar app.jar  #ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"] |

## sb-counter (Java - Spring Boot) 8081. Servicio contador de me gusta.

Spring Boot microservicio para almacenar los me gusta con el id de la foto. Endpoints:

* POST /increment/{id\_foto}
* POST /decrement/{id\_foto}
* GET /{id\_foto}

Se conecta al servicio Redis

* Build imager: docker build -t {nombre\_imagen} .
* Run Container: docker run --name {nombre\_contenedor} -p 8081:8081 {nombre\_imagen}

|  |
| --- |
| **Dockerfile** |
| #jdk extra ligero oficial  FROM openjdk:8-jdk-alpine  VOLUME /tmp  EXPOSE 8081  ADD target/counters-v1.jar app.jar  ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"] |

## sb-photo s(/project) (Node - NestJS) 8082. Servicio de fotos.

Microservice desarrollado con NestJS (TypeScript/NodeJS) que se conecta al servicio MongoDB para almacenar los metadatos y Swift para guarder el archivo. Endpoints:

* Swagger API: /api

\*Swift no se expone publicamente ni genera URL públicas, se hace una consulta a este microservicio que se conecta a Swift y devuelve la foto que se pide. De esta manera se garantiza la privacidad y que ningun usuario no autenticado puede accede a la foto.

* Build imager: docker build -t {nombre\_imagen}.
* Build Container: docker run --name {nombre\_contenedor} -p 8082:8082 {nombre\_imagen}

|  |
| --- |
| **Dockerfile** |
| #node extra ligero oficial  FROM node:carbon-alpine  WORKDIR /nestjs  COPY package\*.json ./  RUN npm install  COPY . ./  EXPOSE 8082  #CMD [ "/bin/sh" ]  ENTRYPOINT ["npm", "run", "start"] |

# Ejecución de docker-compose.

Con docker-compose comprobamos en nuestro docker local que funcionan nuestros microservicios e imágenes.

|  |
| --- |
| **docker-compose.yml** |
| version: "3"  services:  mysql:  image: mysql:5.7 # Imagen oficial mysql del hub de docker  ports:  - "3306:3306" # Exponemos puerto solo para poder debuggear  environment:  - MYSQL\_ROOT\_PASSWORD=root # Contraseña para root  - MYSQL\_DATABASE=users # Crear BD users  login:  build: sb-login # Construir imagen desde carpeta  ports:  - "8080:8080"  environment:  - SECRET\_TOKEN=secreto  - DB\_MYSQL\_DNS=mysql # URL del servicio mysql  - DB\_MYSQL\_USER=root  - DB\_MYSQL\_PW=root  - WAIT\_HOSTS=mysql:3306 # Variables de entorno para esperar a que el servicio mysql esté listo para aceptar querys  - WAIT\_HOSTS\_TIMEOUT=240  swift:  image: twcammaster.uv.es/swift  mongodb:  image: mongo  ports:  - 27017:27017 # Exponemos puerto solo para poder debuggear  photos:  build: project # Construir imagen desde carpeta  ports:  - 8082:8082  environment:s  - SECRET\_TOKEN=secreto  - SWIFT\_URL=http://swift:8080 # URL del servicio swift  - SWIFT\_USERNAME=test:tester  - SWIFT\_PASSWORD=testing  - MONGODB\_URL=mongodb://mongodb/photo  redis:  image: redis  ports:  - 6379:6379 # Exponemos puerto solo para poder debuggear  counters:  build: sb-counters  ports:  - 8081:8081  environment:  - REDIS\_COUNTERS\_SERVICE\_HOST=redis # URL del servicio redis  - SECRET\_TOKEN=secreto |

# Aprovisionamiento de las imágenes en el repositorio (twcammaster.uv.es).

# Por cada una de las tres imágenes ejecutamos:

# $ docker tag {nombre\_imagen} twcammaster.uv.es/{nombre\_imagen}

# $ docker push twcammaster.uv.es/{nombre\_imagen}

# Al realizar el tag si no indicamos la versión la sube como LATEST, la última de nuestro local.

El resto de servicios se obtienen del repositorio Docker-hub (mongodb, redis, mysql) y Swift ya existe en el repository twcammaster.uv.es

# Aprovisionamiento de una instancia con docker-machine.

Creamos una instancia según dictamina el enunciado de la práctica y docker-machime instalado.

docker-machine **--native-ssh** create --openstack-tenant-name proyecto4 --openstack-flavor-name m1.large --openstack-image-name ubuntu-16-aufs --openstack-net-name red-practica3 --openstack-ssh-user ubuntu --openstack-domain-name Default --openstack-floatingip-pool external-network --driver openstack docker-swarm-manager-g4

--native-ssh 🡪 para permitir la conexión ssh desde nuestro sistema operativo local (Windows)

C:\Users\Samuanv>docker-machine --native-ssh create --openstack-tenant-name proyecto4 --openstack-flavor-name m1.large --openstack-image-name ubuntu-16-aufs --openstack-net-name red-practica3 --openstack-ssh-user ubuntu --openstack-domain-name Default --openstack-floatingip-pool external-network --driver openstack docker-swarm-manager-g4

Running pre-create checks...

Creating machine...

(docker-swarm-manager-g4) Creating machine...

Waiting for machine to be running, this may take a few minutes...

Esperamos que esté disponible el ssh

Detecting operating system of created instance...

Waiting for SSH to be available...

Detecting the provisioner...

Provisioning with ubuntu(systemd)...

Installing Docker...

Copying certs to the local machine directory...

Instalación de docker en la instancia

Copying certs to the remote machine...

Setting Docker configuration on the remote daemon...

Checking connection to Docker...

Docker is up and running!

To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env docker-swarm-manager-g4

C:\Users\Samuanv>docker-machine --native-ssh ssh docker-swarm-manager-g4

Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-119-generic x86\_64)

 \* Documentation:  [https://help.ubuntu.com](https://help.ubuntu.com/)

 \* Management:     [https://landscape.canonical.com](https://landscape.canonical.com/)

 \* Support:        <https://ubuntu.com/advantage>

Comprobamos que se instala docker

  Get cloud support with Ubuntu Advantage Cloud Guest:

<http://www.ubuntu.com/business/services/cloud>

41 packages can be updated.

23 updates are security updates.

Last login: Thu Apr 19 10:59:17 2018

]0;ubuntu@docker-swarm-manager-g4: ~ubuntu@docker-swarm-manager-g4:~$ docker- [ [K -vafqterq

unknown shorthand flag: 'a' in -afqterq

See 'docker --help'.

Usage:  docker COMMAND

A self-sufficient runtime for containers

Options:

      --config string         Location of client config files (default

                              "/home/ubuntu/.docker")

  -D, --debug                 Enable debug mode

  -H, --host list             Daemon socket(s) to connect to

  -l, --log-level string      Set the logging level

                              ("debug"|"info"|"warn"|"error"|"fatal")

                              (default "info")

      --orchestrator string   Which orchestrator to use with the docker

                              cli (swarm|kubernetes) (default swarm)

                              (experimental)

      --tls                   Use TLS; implied by --tlsverify

      --tlscacert string      Trust certs signed only by this CA (default

                              "/home/ubuntu/.docker/ca.pem")

      --tlscert string        Path to TLS certificate file (default

                              "/home/ubuntu/.docker/cert.pem")

      --tlskey string         Path to TLS key file (default

                              "/home/ubuntu/.docker/key.pem")

      --tlsverify             Use TLS and verify the remote

  -v, --version               Print version information and quit

Management Commands:

  checkpoint  Manage checkpoints

  config      Manage Docker configs

  container   Manage containers

  image       Manage images

  manifest    Manage Docker image manifests and manifest lists

  network     Manage networks

  node        Manage Swarm nodes

  plugin      Manage plugins

  secret      Manage Docker secrets

  service     Manage services

  stack       Manage Docker stacks

  swarm       Manage Swarm

  system      Manage Docker

  trust       Manage trust on Docker images

  volume      Manage volumes

Commands:

  attach      Attach local standard input, output, and error streams to a running container

  build       Build an image from a Dockerfile

  commit      Create a new image from a container's changes

  cp          Copy files/folders between a container and the local filesystem

  create      Create a new container

  deploy      Deploy a new stack or update an existing stack

  diff        Inspect changes to files or directories on a container's filesystem

  events      Get real time events from the server

  exec        Run a command in a running container

  export      Export a container's filesystem as a tar archive

  history     Show the history of an image

  images      List images

  import      Import the contents from a tarball to create a filesystem image

  info        Display system-wide information

  inspect     Return low-level information on Docker objects

  kill        Kill one or more running containers

  load        Load an image from a tar archive or STDIN

  login       Log in to a Docker registry

  logout      Log out from a Docker registry

  logs        Fetch the logs of a container

  pause       Pause all processes within one or more containers

  port        List port mappings or a specific mapping for the container

  ps          List containers

  pull        Pull an image or a repository from a registry

  push        Push an image or a repository to a registry

  rename      Rename a container

  restart     Restart one or more containers

  rm          Remove one or more containers

  rmi         Remove one or more images

  run         Run a command in a new container

  save        Save one or more images to a tar archive (streamed to STDOUT by default)

  search      Search the Docker Hub for images

  start       Start one or more stopped containers

  stats       Display a live stream of container(s) resource usage statistics

  stop        Stop one or more running containers

  tag         Create a tag TARGET\_IMAGE that refers to SOURCE\_IMAGE

  top         Display the running processes of a container

  unpause     Unpause all processes within one or more containers

  update      Update configuration of one or more containers

  version     Show the Docker version information

  wait        Block until one or more containers stop, then print their exit codes

Run 'docker COMMAND --help' for more information on a command.

]0;ubuntu@docker-swarm-manager-g4: ~ubuntu@docker-swarm-manager-g4:~$ docker-v

docker-v: command not found

]0;ubuntu@docker-swarm-manager-g4: ~ubuntu@docker-swarm-manager-g4:~$ docker -v

Docker version 18.05.0-ce, build f150324

]0;ubuntu@docker-swarm-manager-g4: ~ubuntu@docker-swarm-manager-g4:~$ exit

logout

Como nos indicó en el aprovisionamiento: To see how to connect your Docker Client to the Docker Engine running on this virtual machine, run: docker-machine env docker-swarm-manager-g4. Lo ejecutamos

C:\Users\Samuanv>**docker-machine env docker-swarm-manager-g4**

SET DOCKER\_TLS\_VERIFY=1

SET DOCKER\_HOST=tcp://[147.156.86.12:2376](http://147.156.86.12:2376/)

SET DOCKER\_CERT\_PATH=C:\Users\Samuanv\.docker\machine\machines\docker-swarm-manager-g4

SET DOCKER\_MACHINE\_NAME=docker-swarm-manager-g4

SET COMPOSE\_CONVERT\_WINDOWS\_PATHS=true

REM Run this command to configure your shell:

REM     @FOR /f "tokens=\*" %i IN ('docker-machine env docker-swarm-manager-g4') DO @%i

Contenedores en local

C:\Users\Samuanv>docker ps

CONTAINER ID        IMAGE                         COMMAND                  CREATED             STATUS              PORTS                    NAMES

8692921ed24c        mysql:5.7                     "docker-entrypoint.s…"   9 hours ago         Up 5 hours          0.0.0.0:3306->3306/tcp   mysqlAirline

7726499bbb20        [twcammaster.uv.es/ocata-cli](http://twcammaster.uv.es/ocata-cli)   "/bin/bash"              31 hours ago        Up 5 hours                                   openstack-cli

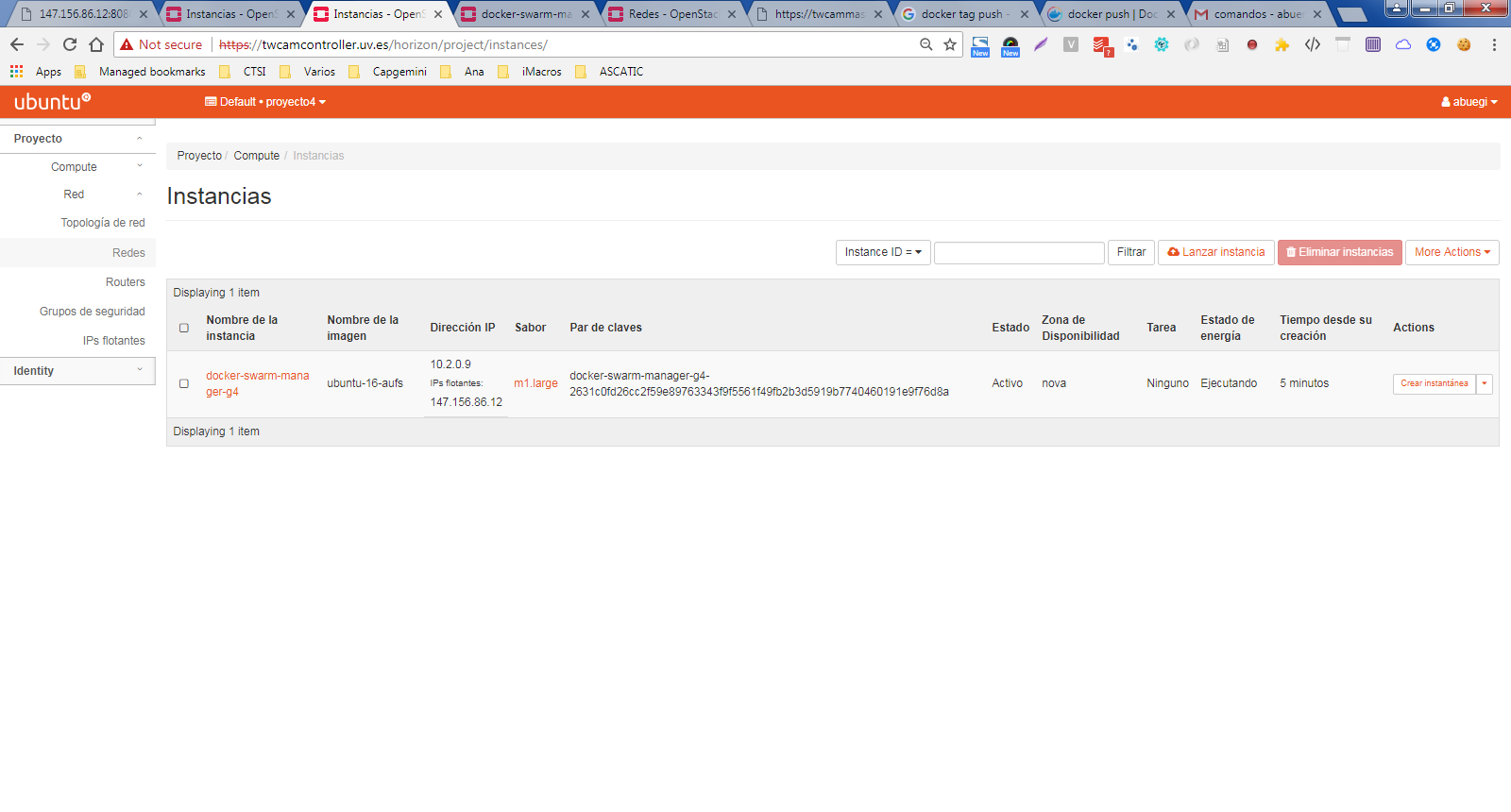
C:\Users\Samuanv**>@FOR /f "tokens=\*" %i IN ('docker-machine env docker-swarm-manager-g4') DO @%i**

C:\Users\Samuanv>docker image list

REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE

Imágenes en remoto. Después de ejecutar: @FOR /f "tokens=\*" %i IN ('docker-machine env docker-swarm-manager-g4') DO @%i

Comprobamos que se ha creado la instancia en openstack



# Definición del fichero YAML con todos los servicios.

Hay que realizar una serie de cambios en el fichero docker-cloud con respecto al docker-compose para dejarlo preparado y que se pueda ejecutar en un cluster.

* Network: Se crea una network overlay para asegurar que sólo se comunican los servicios de este stack.
* Configs: Las variables de entorno de los servicios login y counters son sobreescritas por nuestro fichero application.properties. De esta manera nos ahorramos establecer las variables de entorno directamente en el fichero .yml. Sin embargo, el servicio de photos sigue cogiendo la configuración mediante variables de entorno de forma intencionada, para demostrar las dos formas que hay de realizar esto.
* Images: Todos los servicios se obtienen de images en repository (en docker-compose hacíamos build para desarrollar)

|  |
| --- |
| **docker-cloud.yml** |
| version: "3.3"  services:  mysql:  image: mysql:5.7  environment:  - MYSQL\_ROOT\_PASSWORD=superSecretPW  - MYSQL\_DATABASE=users    networks:  - network\_g4  login:  image: twcammaster.uv.es/login\_g4  ports:  - "8080:8080"  networks:  - network\_g4  configs:  - application.properties  swift:  image: twcammaster.uv.es/swift  networks:  - network\_g4  mongodb:  image: mongo  networks:  - network\_g4  photos:  image: twcammaster.uv.es/photos\_g4  ports:  - 8082:8082  environment:  - SECRET\_TOKEN=secretSwarmSalt  - SWIFT\_URL=http://swift:8080  - SWIFT\_USERNAME=test:tester  - SWIFT\_PASSWORD=testing  - MONGODB\_URL=mongodb://mongodb/photo  networks:  - network\_g4  redis:  image: redis  networks:  - network\_g4  counters:  image: twcammaster.uv.es/counters\_g4  ports:  - 8081:8090  networks:  - network\_g4  configs:  - application.properties  networks:  network\_g4:  configs:  application.properties:  file: application.properties |

# Acciones a realizar.

* 1. **Reiniciar la máquina virtual**

docker-cli (local)

* + docker-machine restart docker-swarm-manager-g4

(utilizar --native-ssh si espera la conexión por ssh)

Borrar el stack (grupo de servicios) si ya existe.

* + docker-machine env docker-swarm-manager-g4

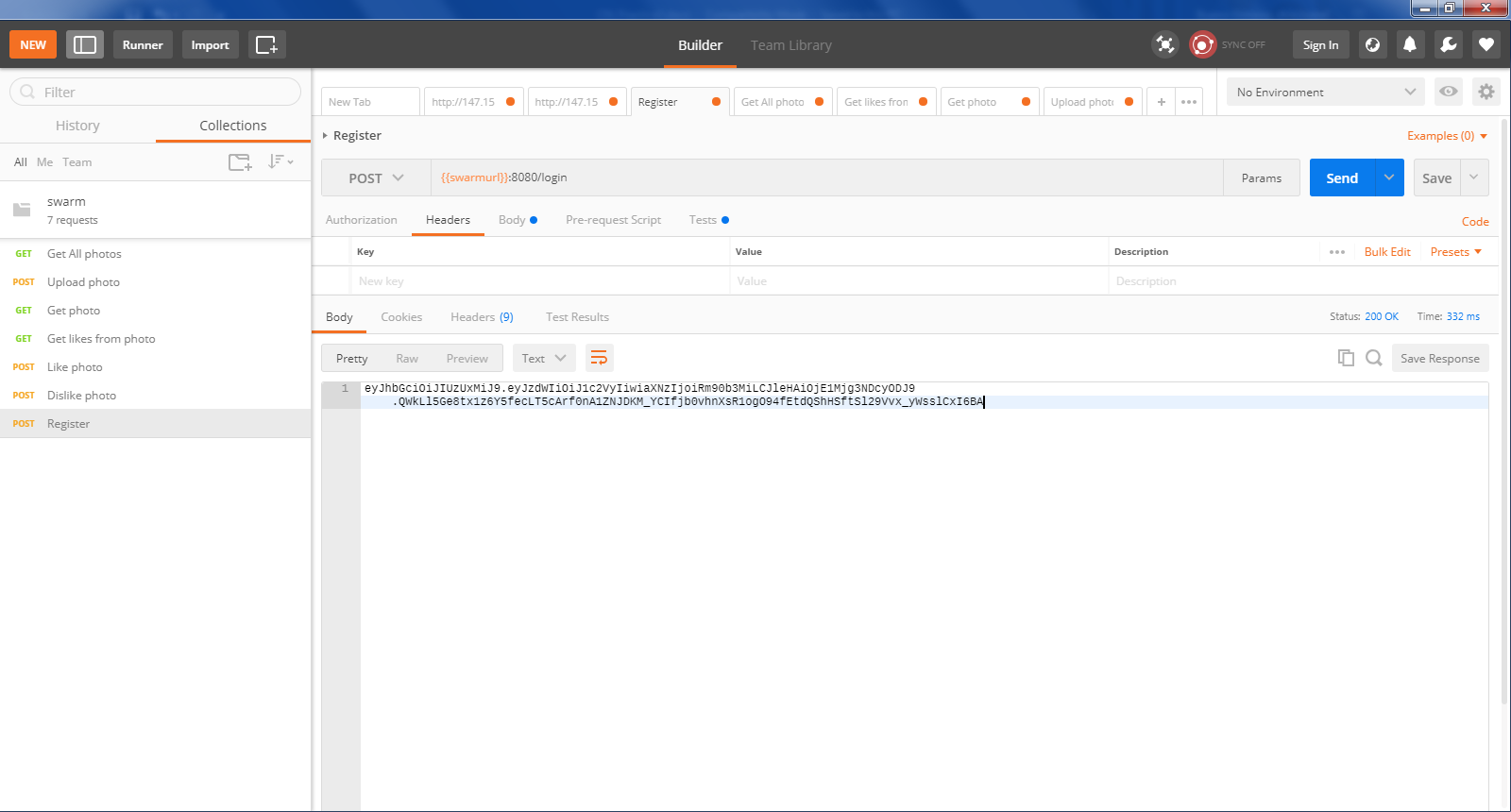
(@FOR /f " ……..

* + docker stack rm g4
  1. **Ejecutar el stack**

docker stack deploy --compose-file docker-cloud.yml

* 1. **Realizar peticiones al servicio**

Junto con el código, se puede encontrar un fichero llamado *swarm.postman\_collection.json* el cual recoge todas las llamadas que se le pueden hacer a los microservicios. Hay que tener en cuenta que hay que tener configurada la variable de entorno *{{swarmurl}}* con el valor de la IP y *{{tokenswarm}}* creada pero vacia. Al realizar el registro o el login, se guardará automáticamente el token



* POST Register

{{swarmurl}}:8080/**register**

Body

{"user":"user",

"password":"pw"}

* POST Register

{{swarmurl}}:8080/**login**

Body:

{"user":"user",

"password":"pw"}

Response:

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJ1c2VyMSIsImlzcyI6IkZvdG9zIiwiZXhwIjoxNTI4NzUyMTY5fQ.ACHx7ueYaSZvHjol45FBrLCnRY6\_q5lHVavwTBXRPDZig4vWCfeDZEsuqRdAwXk7MsF6hmgldau3X4f4SH4Mng

* GET Get All photos

{{swarmurl}}:8082/photo

Headers:

Key: Authorization

value: Bearer {{tokenSwarm}}

* POST Upload photo

{{swarmurl}}:8082/photo

Headers:

Key: Authorization

value: Bearer {{tokenSwarm}}

* GET Get photo

{{swarmurl}}:8082/photo/{photo\_id}

Headers:

Key: Authorization

value: Bearer {{tokenSwarm}}

Key: Content-type

value: application/json

Response:

{image}

* GET Get likes from photo

{{swarmurl}}:8082/photo/{photo\_id}

Headers:

Key: Authorization

value: Bearer {{tokenSwarm}}

* POST Like photo

{{swarmurl}}:8081/increment/{photo\_id}

Headers:

Key: Authorization

value: Bearer {{tokenSwarm}}

* POST Dislike photo

{{swarmurl}}:8081/decrement/{photo\_id}

Headers:

Key: Authorization

value: Bearer {{tokenSwarm}}

* 1. **Escalar el micro-servicio de contadores (de una a dos instancias)**

docker service scale counters =2

* 1. **Detener el stack**

docker stack rm g4

* 1. **Detener la máquina virtual**

docker-cli (local)

* + docker-machine stop docker-swarm-manager-g4

(utilizar --native-ssh si espera la conexión por ssh)

## Extras:

### Portainer

Stack que se levanta en el nodo Manager con el fin de poder gestionar de forma visual el estado de nuestro Cluster. Se despliega mediante la siguiente instrucción (cogida de la documentación oficial de portainer):

Docker stack deploy –compose-file portainer-agent-stack.yml portainer

version: '3'

services:

agent:

image: portainer/agent

environment:

# REQUIRED: Should be equal to the service name prefixed by "tasks." when

# deployed inside an overlay network

AGENT\_CLUSTER\_ADDR: tasks.agent

# AGENT\_PORT: 9001

# LOG\_LEVEL: debug

volumes:

- /var/run/docker.sock:/var/run/docker.sock

networks:

- agent\_network

deploy:

mode: global

portainer:

image: portainer/portainer

command: -H tcp://tasks.agent:9001 --tlsskipverify

ports:

- "9000:9000"

volumes:

- portainer\_data:/data

networks:

- agent\_network

deploy:

mode: replicated

replicas: 1

placement:

constraints: [node.role == manager]

networks:

agent\_network:

driver: overlay

volumes:

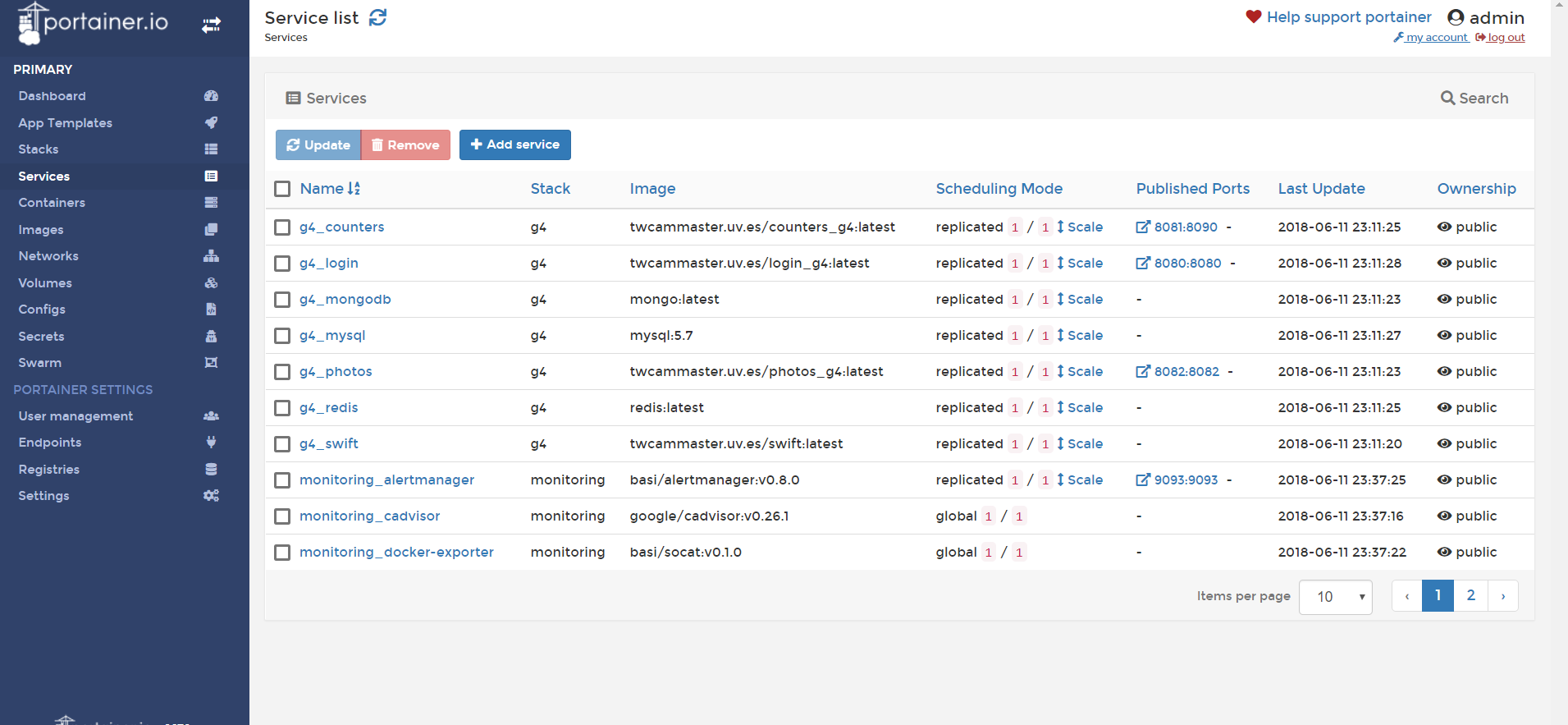
portainer\_data:

Esto nos expone un servicio en el puerto :9000, el cual podemos configurar con usuario y contraseña y diferentes permisos.

En nuestro caso será:

admin

12345678



### Prometheus con Graphana

Podemos instalar Prometheus en cada nodo con la finalidad de recoger métricas de los distintos servicios y poder tener unos logs muy útiles de nuestro cluster.

Podemos ejecutar el siguiente stack:

version: "3"

networks:

g4\_network\_g4:

services:

cadvisor:

image: google/cadvisor:${CADVISOR\_VERSION:-v0.26.1}

networks:

- g4\_network\_g4

volumes:

- /var/run/docker.sock:/var/run/docker.sock,readonly

- /:/rootfs

- /var/run:/var/run

- /sys:/sys

- /var/lib/docker/:/var/lib/docker

deploy:

mode: global

resources:

limits:

cpus: '0.10'

memory: 128M

reservations:

cpus: '0.10'

memory: 64M

node-exporter:

image: basi/node-exporter:${NODE\_EXPORTER\_VERSION:-latest}

networks:

- g4\_network\_g4

volumes:

- /proc:/host/proc

- /sys:/host/sys

- /:/rootfs

- /etc/hostname:/etc/host\_hostname

hostname: "{{.Node.ID}}"

environment:

HOST\_HOSTNAME: /etc/host\_hostname

command: -collector.procfs "/host/proc" -collector.sysfs /host/sys -collector.textfile.directory /etc/node-exporter/ -collectors.enabled 'conntrack,diskstats,entropy,filefd,filesystem,loadavg,mdadm,meminfo,netdev,netstat,stat,textfile,time,vmstat,ipvs' -collector.filesystem.ignored-mount-points "^/(sys|proc|dev|host|etc)($$|/)"

deploy:

mode: global

resources:

limits:

cpus: '0.10'

memory: 32M

reservations:

cpus: '0.10'

memory: 16M

docker-exporter:

image: basi/socat:${DOCKER\_EXPORTER\_VERSION:-v0.1.0}

networks:

- g4\_network\_g4

deploy:

mode: global

resources:

limits:

cpus: '0.05'

memory: 6M

reservations:

cpus: '0.05'

memory: 4M

alertmanager:

image: basi/alertmanager:${ALERTMANAGER\_VERSION:-v0.8.0}

networks:

- g4\_network\_g4

# - logging

ports:

- "9093:9093"

environment:

SLACK\_API: ${SLACK\_API:-YOURTOKENGOESHERE}

LOGSTASH\_URL: http://logstash:8080/

command: -config.file=/etc/alertmanager/config.yml

deploy:

mode: replicated

replicas: 1

resources:

limits:

cpus: '0.01'

memory: 32M

reservations:

cpus: '0.01'

memory: 16M

prometheus:

image: basi/prometheus-swarm:${PROMETHEUS\_SWARM\_VERSION:-latest}

ports:

- "9090"

networks:

- g4\_network\_g4

command: -config.file=/etc/prometheus/prometheus.yml -storage.local.path=/prometheus -web.console.libraries=/etc/prometheus/console\_libraries -web.console.templates=/etc/prometheus/consoles -alertmanager.url=http://alertmanager:9093

deploy:

mode: replicated

replicas: 1

resources:

limits:

cpus: '0.50'

memory: 1024M

reservations:

cpus: '0.50'

memory: 128M

grafana:

image: basi/grafana:${GRAFANA\_VERSION:-4.4.3}

ports:

- "3000:3000"

networks:

- g4\_network\_g4

environment:

GF\_SECURITY\_ADMIN\_PASSWORD: ${GF\_PASSWORD:-admin}

PROMETHEUS\_ENDPOINT: http://prometheus:9090

ELASTICSEARCH\_ENDPOINT: ${ES\_ADDRESS:-http://elasticsearch:9200}

ELASTICSEARCH\_USER: ${ES\_USERNAME}

ELASTICSEARCH\_PASSWORD: ${ES\_PASSWORD}

deploy:

mode: replicated

replicas: 1

resources:

limits:

cpus: '0.50'

memory: 64M

reservations:

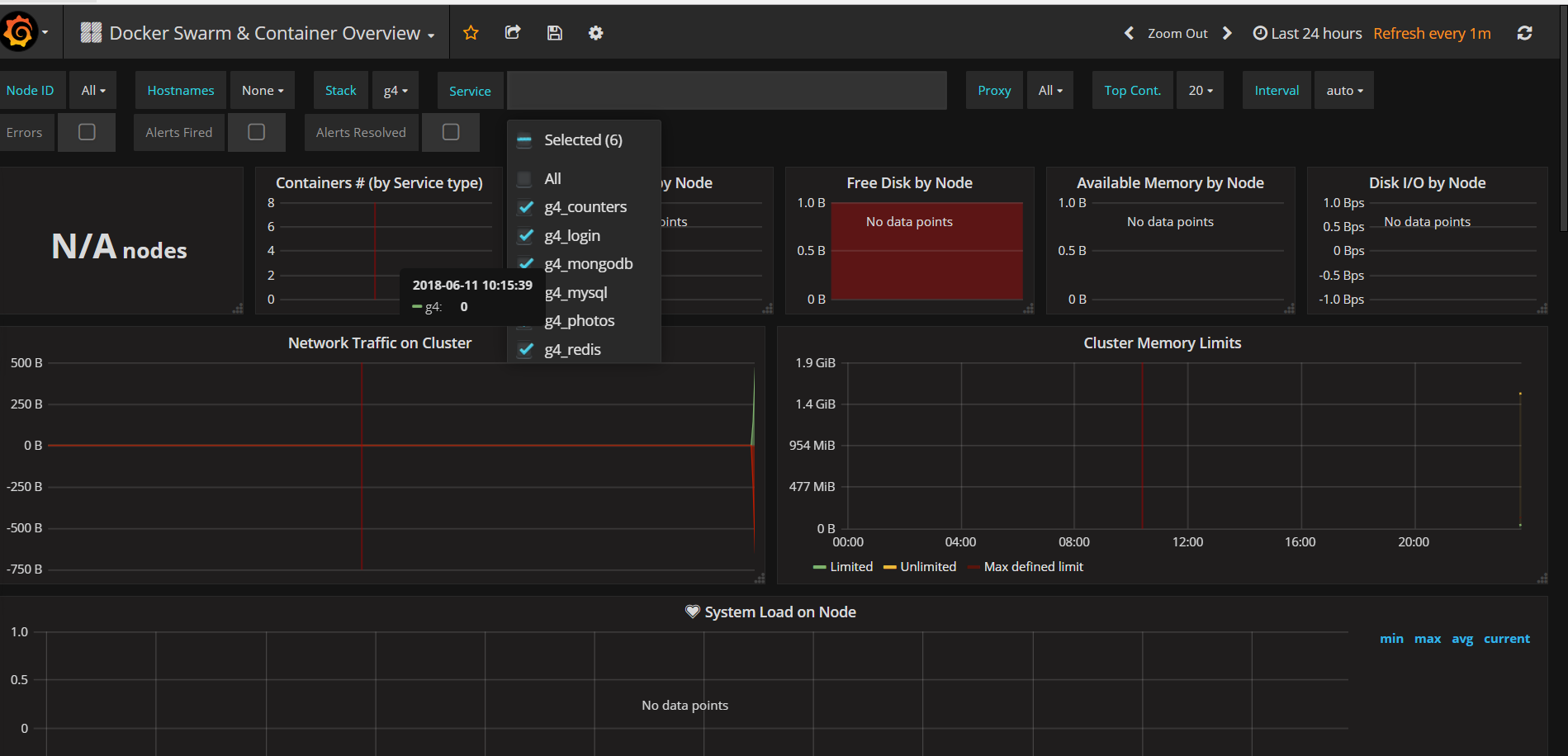
cpus: '0.50'

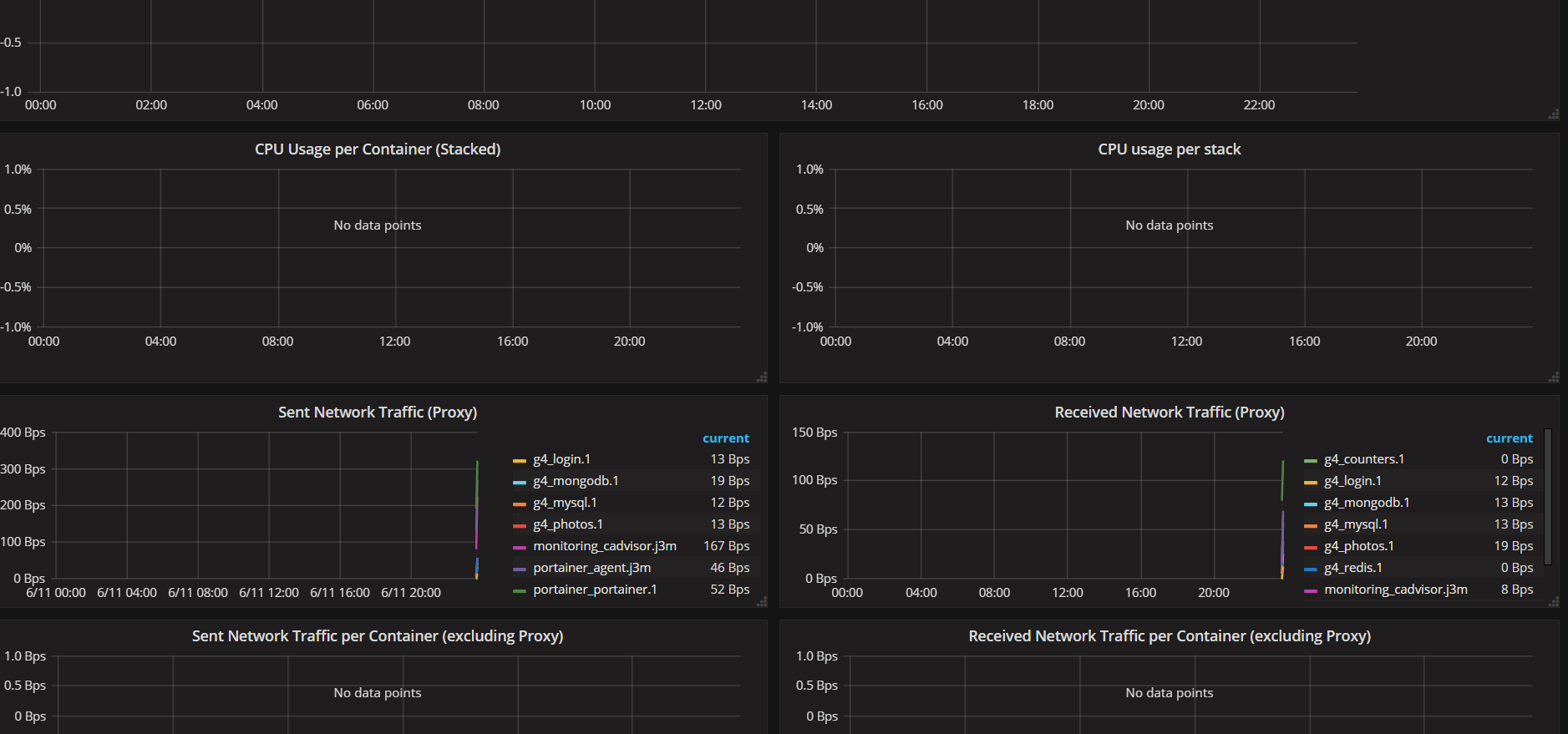
memory: 32M

(https://github.com/bvis/docker-prometheus-swarm)

\*\*\*Nótese que le asignamos la misma red que el otro stack (g4\_network\_g4) para que sea capaz de recoger todos los datos de los contenedores.

Por último accedemos al puerto :3000 e importamos el dashboard de Graphana 609 (<https://grafana.com/dashboards/609>). Con esto ya tenemos un potente panel de control monitorizando todos nuestros servicios.





A esta hora no existe aún tráfico, esperemos al día de la presentación para ver la actividad de nuestros ciberamigos Chinos y Rusos.