



# Docker Workshop - TACS



# Docker run

**docker run**

dockerFile

docker-compose

— — —

# Docker lab

— — —

labs.play-with-docker.com

AccessAir

# Docker run

---

Sintaxis

```
docker run [options] [image] [commands] [args]
```

Ejecutar:

- `docker run ubuntu:14.04 echo "Hello World"`
- `docker run ubuntu ps aux`
- `docker run -it ubuntu:14.04 /bin/bash`

# Container ID

---

- Los containers se pueden especificar por **id** o por **name**
- **longId** & **shortId**
- **shortId** y **name** se obtienen con **docker ps**
- **longId** se obtienen con **docker inspect [name]**

# Detached mode y Logs

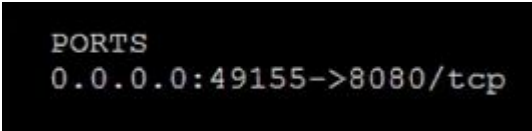
— — —

- `docker run -d centos:7 ping 127.0.0.1 -c 100`
- `docker ps -a`
- `docker logs -f [containerId]`
- `docker rm $(docker ps -qa)`

# Detached mode y Logs

— — —

- `docker run -d -P tomcat:7`
- `docker ps`



PORTS  
0.0.0.0:49155->8080/tcp

- Browser en host:port (ej. *localhost:49155*) o en el dns *provisto por play-with-docker.com*

bmmqpv6d\_bmmr0oudo98g00c558h0

IP

192.168.0.53

32768

- `docker run -d -p 8080:8080 tomcat:7`
- Browser en host:8080 (ej. *localhost:8080*)



# Conceptos de docker

**image**



# Docker Images

---

- **Templates read-only** desde los que se crean containers
- Derivados de una **imagen base**(BusyBox, Alpine o Ubuntu)
- Armada por un **conjunto de capas** que combinadas, conforman el filesystem
- **Construida** a partir de un *Dockerfile*
- Cada capa es **hasheada** y **cacheada**

```
FROM ubuntu:16.10

RUN apt-get update && apt-get install -y apache2

ENTRYPOINT [ "apache2ctl" ]
```

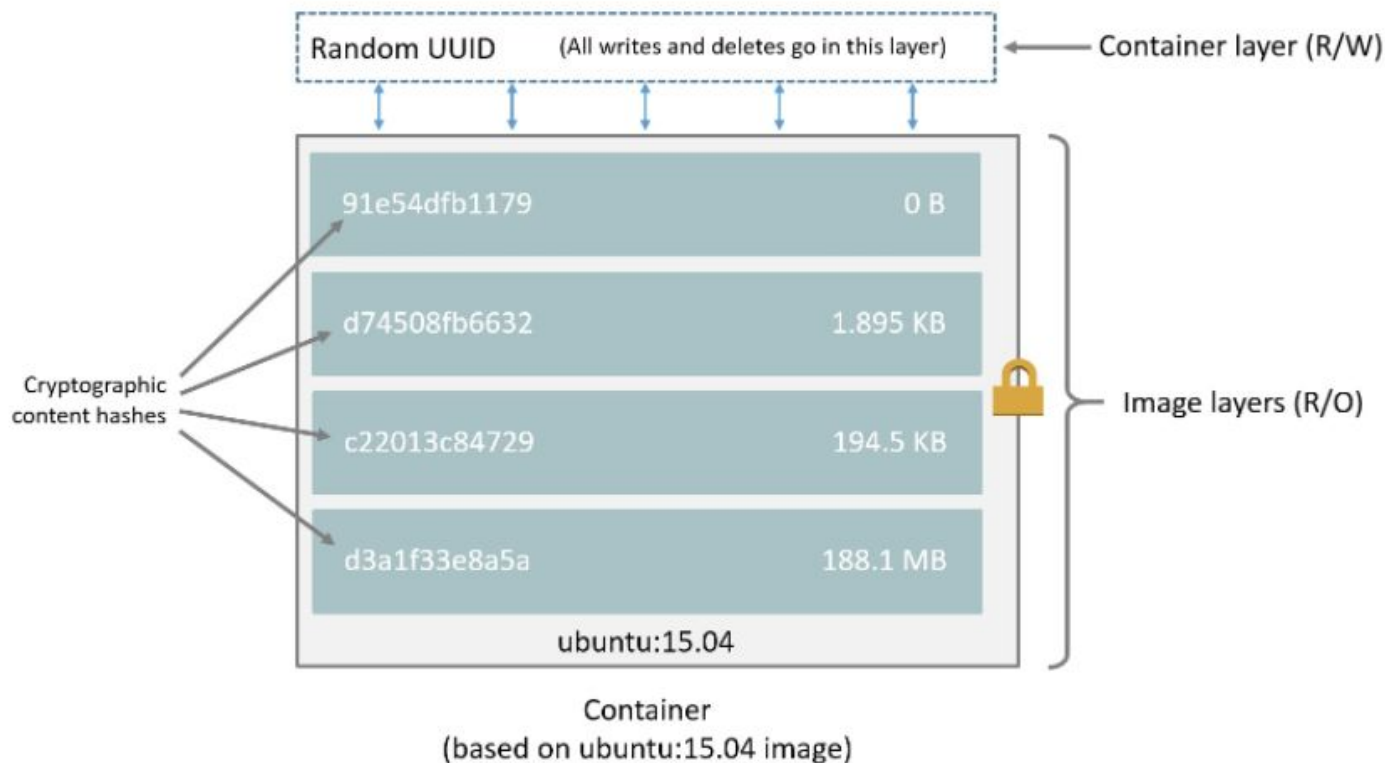
# Docker Images

---

Cada **imagen docker** referencia una **lista de capas read-only** que representan deltas de filesystem.

Estas capas son apiladas una sobre otra, sirviendo como base para formar el **sistema de archivos raíz** de un **container**.

# Docker File System Layers





# Conceptos de docker

**container**

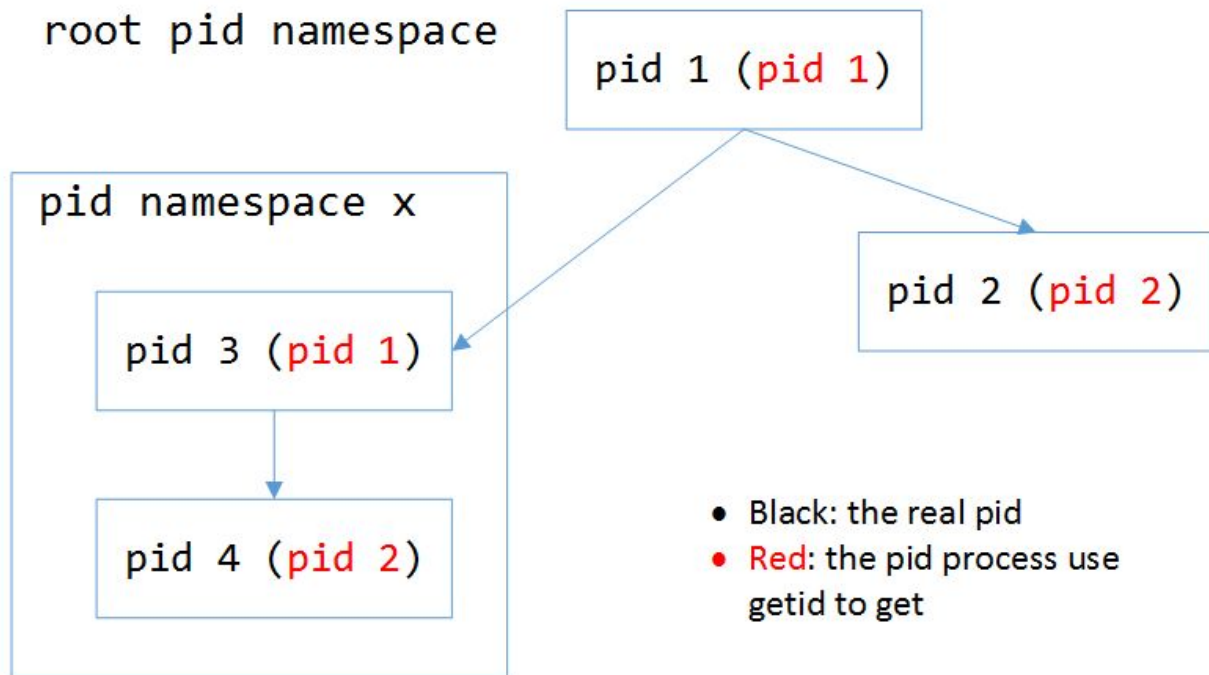
# Docker container

— — —

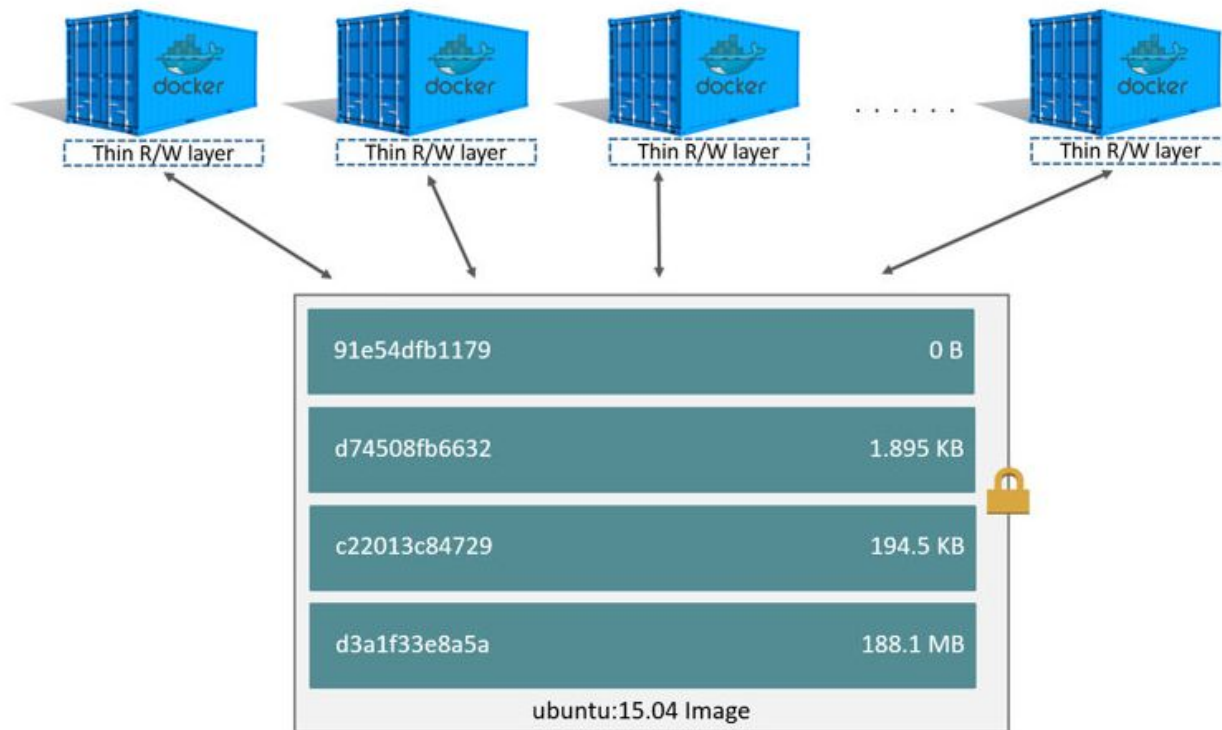
- **instancia runtime** de una **docker image**
- **capa r/w** agregada **arriba** de la imagen
- **efímero**
- **aislado** del resto
  - no puede ver/modificar/dañar el host u otros containers
  - **namespaces**, con los que logran tener su vista privada del sistema (network interfaces, árbol PID, mountpoints..)
  - **cgroups**, para tener recursos limitados, y mitigar el efecto ***bad neighbor***

# namespaces

---



# Docker Containers / FileSystem Layers



# Container y layers

— — —

- La mayor diferencia es la capa R/W que tiene el container
- Capa efímera. Al eliminar un container:
  - Esta capa R/W se elimina
  - La imagen queda sin cambios
- Múltiples containers pueden compartir la misma imagen





# docker commit

docker run

**docker commit**

Dockerfile

docker-compose

— — —

# docker commit

- guarda los cambios de un container en una imagen nueva

— — —

- Sintaxis:

- `docker commit [options] [container ID] [repository:tag]`
- **repository** -> **username/application**

- Probar:

- **`docker run -it ubuntu:14.04 /bin/bash`**
- `curl 127.0.0.1`
- **`apt-get update`**
- **`apt-get install curl`**
- **`rm -rf /var/lib...`**
- **`exit container`**
- `docker ps -a`
- **`docker commit [container ID] tacs-utn/curl:1.0`**
- `docker rm [container ID]`
- `docker run -it tacs-utn/curl:1.0 /bin/bash`
- `curl wttr.in/CABA?m`



# Dockerfile

`docker run`

`docker commit`

**Dockerfile**

`docker-compose`

— — —

# build

— — —

- Dockerfile
- cada **comando** es una **nueva capa** (hace un **commit** por comando)
- **capas jerárquicas**: cambiar una capa baja, destruye el cache de las capas superiores
- **download**: todo esto se descargará una y otra vez
- image tag (uno o más por imagen)

# Dockerfile - Best Practices

— — —

- Containers deben ser efímeros
- Reducir tamaño de las imagenes:
  - Usar un archivo `.dockerignore` para archivos innecesarios en runtime
  - Evitar instalar paquetes innecesarios
  - Cada container un concern
  - Minimizar la cantidad de **capas**

# Dockerfile - docker build command

---

- Sintaxis
- `docker build [options] [path]`
- `docker build -t [repository:tag] [path]`

# Dockerfile - docker build command

---

- Crear una carpeta test y un archivo Dockerfile
  - `mkdir test`
  - `cd test`
  - `vim Dockerfile`
- Editar Dockerfile:

```
FROM ubuntu:14.04
```

```
RUN apt-get update && apt-get install -y curl && rm -rf /var/lib/apt/lists/
```

- `docker build -t tacs-utn/curluntu:1.0 .`
- crear un container y verificar que tiene curl instalado

# Dockerfile - CMD

---

- Define un comando a ser ejecutado al crear un container
- Puede ser redefinido **en runtime**
- No hace nada durante el build
- Editar Dockerfile:

```
FROM ubuntu:14.04
```

```
RUN apt-get update && apt-get install -y curl
```

```
CMD ping 127.0.0.1 -c 30
```

- `docker build -t tacs-utn/pinger:1.0 .`
- `docker run tacs-utn/pinger:1.0`
- `docker run tacs-utn/pinger:1.0 echo "hello world"`



# Dockerfile - ENTRYPOINT

— — —

- Define un comando a ser ejecutado al crear un container
- Se pasan los parámetros **en runtime**
- No hace nada durante el build
- Editar Dockerfile:

```
FROM ubuntu:14.04
```

```
RUN apt-get update && apt-get install -y curl
```

```
ENTRYPOINT ["ping"]
```

- `docker build -t tacs-utn/pinger:1.1 .`
- `docker run tacs-utn/pinger:1.1`
- `docker run tacs-utn/pinger:1.1 127.0.0.1 -c 5`

# docker start / stop

— — —

- Arranca o detiene containers creados
- Ejecutar:
  - `docker run -d nginx`
  - `docker ps`
  - `docker stop [container ID]`
  - `docker ps -a`
  - `docker start [container ID or name]`

# Docker exec - Acceso via terminal a un container vivo

— — —

- Arranca un proceso dentro de un container
- Ejecutar:
  - `docker run -d nginx`
  - `docker ps`
  - `docker exec -it [container ID] /bin/bash`

# Data Volumes

---

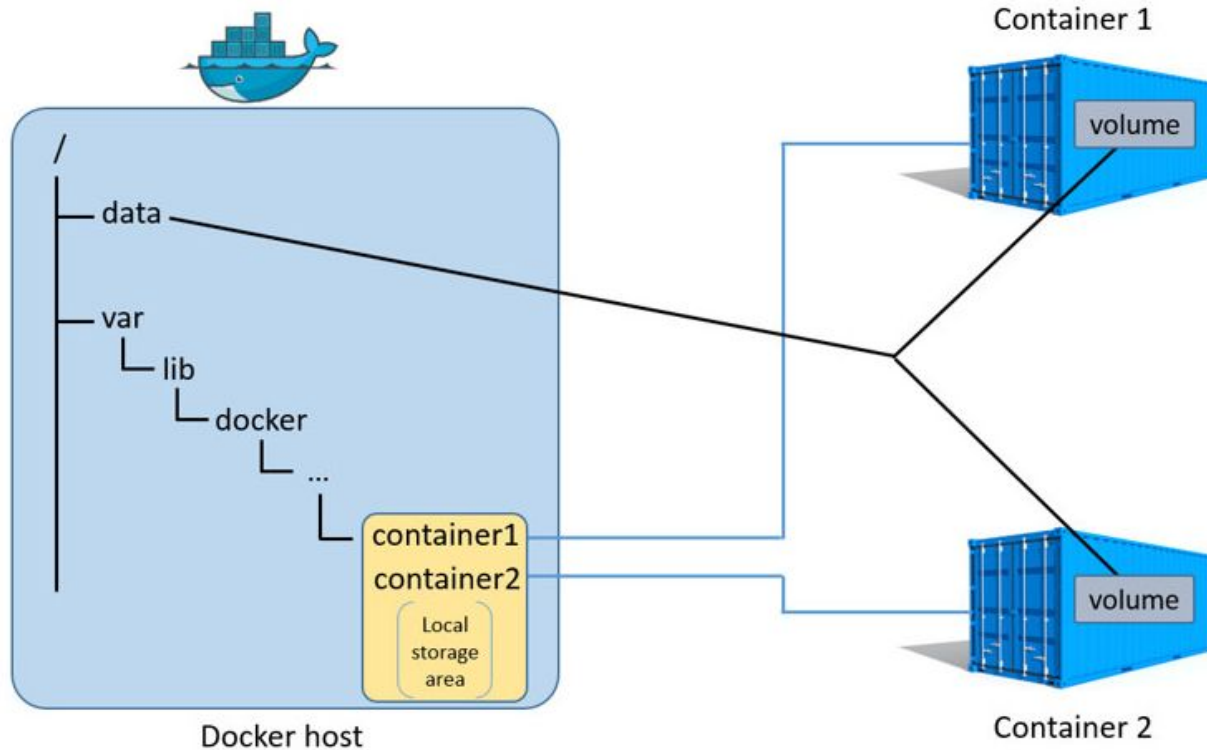
Cuando se borra un container, cualquier dato escrito en el container, que no es almacenado en un **data volume**, es **eliminado junto con el container**.

# Data Volumes

---

- **Directorio o archivo** en el filesystem del **host**, **montado** directamente dentro de un container.
- **No** son controlados por el **storage driver**
- Operan a la **velocidad nativa** del host
- Se puede montar cualquier número de **data volumes** en un container
- **Multiples containers** pueden **compartir** uno o más **data volumes**

# Data Volumes





# docker-compose

docker run

docker commit

Dockerfile

**docker-compose**

— — —

# Docker compose

---

Herramienta que nos permite definir y correr aplicaciones docker multi containers

Nos permite definir su arquitectura mediante código:

- services
  - building
  - links
  - ports
- volumes
- networks



# Docker compose

---

```
docker-compose-network.yaml
1  version: "3.3"
2  services:
3    gateway:
4      container_name: gateway
5      image: nginx/unit:1.23.0-go1.15
6      ports:
7        - 80:8080
8      networks:
9        - frontend
10   backend-for-frontend:
11     container_name: backend-for-frontend
```

# Docker compose

— — —

- git clone <https://github.com/tacs-utn/docker-networking.git>
- **docker compose up -d**
- docker exec **gateway** curl backend-for-frontend:8080
- docker exec **gateway** curl backend:8080
- docker exec **gateway** curl database:8080
- **check ips:**
  - docker exec **gateway** ip r
  - docker exec **backend-for-frontend** ip r
  - docker exec **backend** ip r
  - docker exec **database** ip r

# Docker compose

— — —

- git clone <https://github.com/tacs-utn/docker-networking.git>
- **docker compose** -f *docker-compose-network.yaml* **up** -d
- docker exec **gateway** curl backend-for-frontend:8080
- docker exec **gateway** curl backend:8080
- docker exec **gateway** curl database:8080
- **check ips:**
  - docker exec **gateway** ip r
  - docker exec **backend-for-frontend** ip r
  - docker exec **backend** ip r
  - docker exec **database** ip r

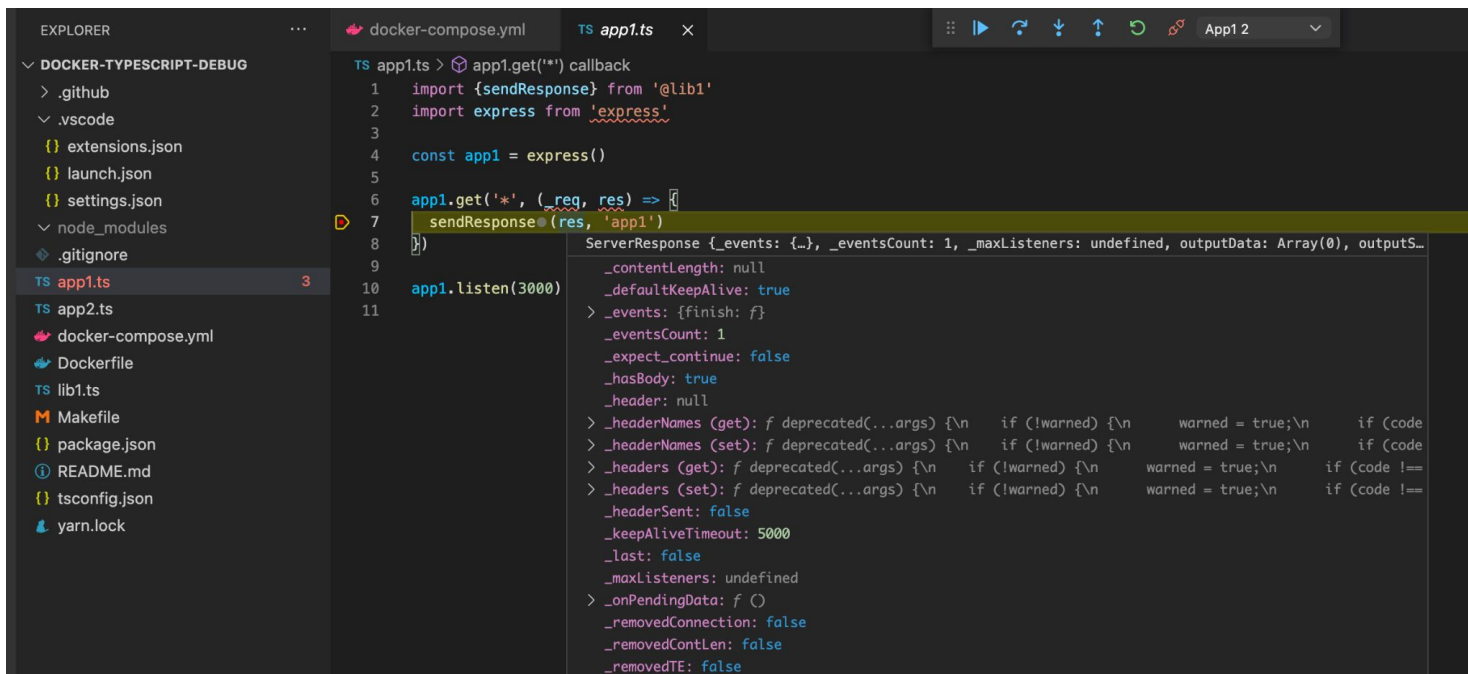
# Docker compose

— — —

- `git clone https://github.com/tacs-utn/docker-networking.git`
- `docker compose -f docker-compose-network.yaml up -d`
- `docker compose scale backend-for-frontend=5 backend=3`
- **check different ips on service discovery balancing:**
  - *(repeat x5)* `docker exec gateway ping backend-for-frontend -c 1`
  - *(repeat x3)* `docker exec docker-networking_backend-for-frontend_1 ping backend -c 1`

# Debugging Node.js with Hot Reload

- <https://github.com/alejandropal/docker-typescript-debug>
- <https://www.youtube.com/watch?v=1WUoITRINf0>



The screenshot shows the VS Code interface with a project named 'DOCKER-TYPESCRIPT-DEBUG'. The Explorer sidebar on the left lists files including .github, .vscode, extensions.json, launch.json, settings.json, node\_modules, .gitignore, app1.ts (selected), app2.ts, docker-compose.yml, Dockerfile, lib1.ts, Makefile, package.json, README.md, tsconfig.json, and yarn.lock. The main editor displays the code for app1.ts, which imports sendResponse from @lib1 and express from 'express', creates an express app, and sets up a GET route for '\*' that calls sendResponse. The app is listening on port 3000. A yellow tooltip is visible over the sendResponse call on line 7, showing a detailed error message: 'ServerResponse {events: {...}, \_eventsCount: 1, \_maxListeners: undefined, outputData: Array(0), outputS...'. The error details include properties like \_contentLength, \_defaultKeepAlive, \_events, \_eventsCount, \_expect\_continue, \_hasBody, \_header, \_headerNames, \_headers, \_headerSent, \_keepAliveTimeout, \_last, \_maxListeners, \_onPendingData, \_removedConnection, \_removedContLen, and \_removedTE.

```
TS app1.ts > app1.get('*') callback
1  import {sendResponse} from '@lib1'
2  import express from 'express'
3
4  const app1 = express()
5
6  app1.get('*', (_req, res) => {
7    sendResponse(res, 'app1')
8  })
9
10 app1.listen(3000)
11
```

ServerResponse {events: {...}, \_eventsCount: 1, \_maxListeners: undefined, outputData: Array(0), outputS...

\_contentLength: null  
\_defaultKeepAlive: true  
> \_events: {finish: f}  
\_eventsCount: 1  
\_expect\_continue: false  
\_hasBody: true  
\_header: null  
> \_headerNames (get): f deprecated(...args) {\n if (!warned) {\n warned = true;\n if (code  
> \_headerNames (set): f deprecated(...args) {\n if (!warned) {\n warned = true;\n if (code  
> \_headers (get): f deprecated(...args) {\n if (!warned) {\n warned = true;\n if (code !=  
> \_headers (set): f deprecated(...args) {\n if (!warned) {\n warned = true;\n if (code !=  
\_headerSent: false  
\_keepAliveTimeout: 5000  
\_last: false  
\_maxListeners: undefined  
> \_onPendingData: f ()  
\_removedConnection: false  
\_removedContLen: false  
\_removedTE: false

# Gracias!

TACS

# Referencias

— — —

- <https://docs.docker.com/>
- <https://www.slideshare.net/TriNimbus/from-your-desktop-to-the-cloud-things-you-need-to-consider-before-you-run-docker-in-aws>
- [https://github.com/DataDog/the-monitor/blob/master/docker/1 the docker monitoring problem.md](https://github.com/DataDog/the-monitor/blob/master/docker/1%20the%20docker%20monitoring%20problem.md)
- <https://docs.docker.com/get-started/resources/>
- <http://www.dwmkerr.com/learn-docker-by-building-a-microservice>