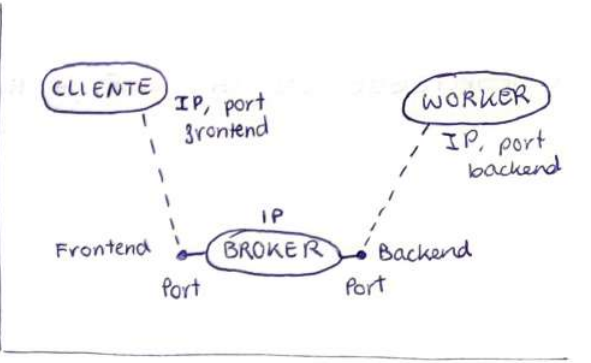
TEMA 4

- **DESPLIEGUE** → Instalación, Activación, Actualización, Eliminación } Componentes => Software preparado para su uso

- **TENER EN CUENTA**
  - Componentes heterogeneos en red
    - creados por desarrolladores distintos
    - cambiantes
    - ↓
    - Deben cooperar → dependencias
    - Nodos heterogeneos → requisitos
    - Seguridad



- **EJEMPLO → PATRÓN BROKER**
  - Cliente - Broker - Worker => autónomos
  - Varias instancias de cada uno => independientes
  - CLIENTE < ubicación del broker (IP y puerto frontend)
  - WORKER < ubicación del broker (IP y puerto backend)
  - } Posee una id única cada instancia

- **PROBLEMA** → Despliegue manual (no) es viable para un servicio grande real

Solución ↓

- **DESPLIEGUE AUTOMATIZADO**
  - Aplicación + Despliegue = SERVICIO
    - Establece un SLA (Service Legal Agreement) < Función, Rendimiento, Disponibilidad
  - Desplegar un servicio => INSTALACIÓN, ACTIVACIÓN, ACTUALIZACIÓN, ADAPTACIÓN

PLAN DE DESPLIEGUE ↓

1. CONFIGURACIÓN PARA CADA COMPONENTE
   - Fichero con la configuración + descripción de dependencias
   - La herramienta generará una config. especifica para cada instancia del componente (worker, broker...)
2. PLAN DE CONFIGURACIÓN GLOBAL
   - conexión entre componentes
   - Donde colocar cada instancia
   - Enlace (bindings) de dependencias, endpoints

¿Cómo se resuelven?

1. Mediante el código < Bajo nivel, Propenso a errores
2. Inyección de dependencias – La aplicación define qué recursos necesita pero no cómo obtenerlos; El contenedor se encargará

- DESPLIEGUE EN LA NUBE → Hemos creado y seleccionado componentes → ¿Proveedor? < Iaas / Pass

  - Iaas
    - Infraestructura (Hardware)
    - Virtualización (Maquinas Virtuales) → Flexibilidad asignación recursos
    - Limitaciones → Decisiones asignación bajo nivel → especificar 100% todo
      → No permite elegir características
      → Baja recuperación tras fallo

  - PaaS
    - Plataforma
    - SLA como elemento central → Idóneo → Automatización del Despliegue
    - Automatización / Configuración solucionados
    - Establece automaticamente los nodos necesarios

- CONTENEDORES

  - Simplifica el enlace entre dependencias y endpoints
  - Aprovisionamiento → Reservar infraestructura para una apl. distribuida < Recursos de intercomunicación / Recursos para cada instancia
    - Maquina Virtual: SO + Bibliotecas
    - Contenedor: Bibliotecas / El SO del anfitrión
  - Inconvenientes
    - Menor flexibilidad → Compatible con el SO anfitrión
    - Menor protección
  - Ventajas
    - Más ligero → Menos recursos
    - Más rápido
    - Más facil de instalar

↓

DOCKER → Automatiza el despliegue de cada instancia < Inicio / Parada / Eliminación ...

Define un sistema de ficheros nativo, para compartir cosas entre contenedores

- DOCKER
  - IMAGEN
    - Plantilla SOLO lectura
    - Conjunto instrucciones para crear un contenedor
      - En el depósito existen imagenes predefinidas
      - Nueva imagen = imagen base + instrucciones
  - CONTENEDOR
    - Cuando ejecutamos el software descrito en la imagen
    - Recursos que una instancia necesita para ejecutarse
    - Máquina independiente con su propio contexto ← ejecutables, bibliotecas, aplicaciones
  - DEPÓSITO
    - Donde se almacenan las imagenes (local / nube)
    - Para compartir imágenes

- ÓRDENES CONSOLA
  - Estructura → docker acción opciones argumentos
  - CREAR CONTENEDOR →
    - docker run opciones imagen progInicial
    - docker run -i -t centos bash

    -i -t → modo interactivo con la consola

    Descarga la imagen 'centos', crea el contenedor, reserva el sistema de ficheros ... y ejecuta bash
  - CREAR NUEVA IMAGEN → *
    - Con la consola podemos modificar el contenedor
    - Guardar cambios docker commit nombreContenedor nombreImagen

- CREAR NUEVAS IMÁGENES
  - La nueva imagen debe incluir → bibliotecas + interprete + programa a ejecutar
  - Formas: 1° imagen base en disco
    - Interactivamente *
    - DockerFile → Archivo de configuración
      - En la 1ª linea se especifica QUE IMAGEN base se va a modificar (FROM image)
      - Seguido de las mismas lineas de la consola
      - Preparado → docker build -t tsr-zmq
    - Id del contenedor → docker ps -a

- FICHERO DOCKERFILE

  - **FROM** : Nombre de la imagen base a modificar, si no es ninguna → 'latest'
  - **RUN** orden : Ejecuta dicha orden en el shell
  - **ADD** origen destino : Copia ficheros del host al contenedor, add descomprime, sino se usa COPY
  - **EXPOSE** puerto : Indica el puerto en el que el contenedor atenderá peticiones
  - **ENV** variable valor : Establece una variable de entorno accesible por los programas del contenedor
  - **CMD** orden arg1 arg2 : Valores por defecto para la ejecución del contenedor
    - **ENTRYPOINT** orden arg1 arg2 : Ejecuta la orden al crear el contenedor

    Solo habrá una orden CMD o ENTRYPOINT → sirven para lanzar el programa
  - **WORKDIR** path : Directorio de trabajo para las intrucciones RUN, CMD, ENTRYPOINT etc.

- MULTIPLES COMPONENTES

  Queremos lanzar varios componentes (worker, broker, clientes) → hay que iniciar 1º broker → Obtener ID y puertos → modificar Dockerfiles de w y c → COSTOSO

  SOLUCIÓN → DOCKER COMPOSE → Crea un plan de trabajo que describe componentes y relaciones. Las dependencias se resuelven en ejecución

↓

DOCKER COMPOSE

Resuelve las dependencias de componentes externos.

↓

DESPLIEGUE AUTOMATIZADO

↓

Docker-Files Configurables

DOCKER-FILE → se lanza con → docker-compose up -d

docker-compose up -d --scale X=n // para lanzar 'n' instancias (cli / wor)

```
cli:
  image : client → que imagen necesita
  build : ./client/ → si la imagen NO existe se construye
  links :
    - bro → depende del broker, 1º se lanza bro, luego cli y wor, es igual a la IP
  environment
    - BROKER_URL = ...
        ↑ valor de la variable de entorno
```

```
bro:
  image : broker
  build : ./broker/
  expose :
    - "9999"   } en que puertos
    - "9998"   } está escuchando
```

BROKER

```
FROM ....
COPY ....
EXPOSE 9999 9998
CMD node broker
```

WORKER

```
FROM ...
COPY ...
CMD node worker $BROKER_URL
```

CLIENTE

```
FROM ...
COPY ...
CMD node client $BROKER_URL
```

↓

Variable de entorno, se le da valor en el DockerFile

- MULTIPLES COMPONENTES EN DISTINTOS NODOS

Utilizar PaaS o clústers

Kubernetes → distribuir instancias entre distintos nodos, nada que ver con Docker

↓

Distribuidor de contenedores

Se compone de nodos virtuales o físicos + 'pods' ← pequeñas unidades para el despliegue

- A TENER EN CUENTA

EXPOSE port : Indica que un contenedor escuchará en un puerto específico

↑

SI + de uno igual

Aunque varios contenedores se lancen en el mismo nodo, cada uno tiene sus puertos.

Si varios lanzan 'EXPOSE 8000' cada uno tiene su propio puerto 8000.

PORTS port : Conecta los puertos en los que un contenedor está escuchando a un puerto real de la máquina.

↑

NO + de uno igual

Aquí NO puede usar dos veces el mismo puerto. Ya que solo hay un puerto real.