Virtualización y contenedores

Unidad 3

Contenidos

- Volúmenes
- Network
- Docker Compose
- Kubernetes

Volúmenes

Volúmenes

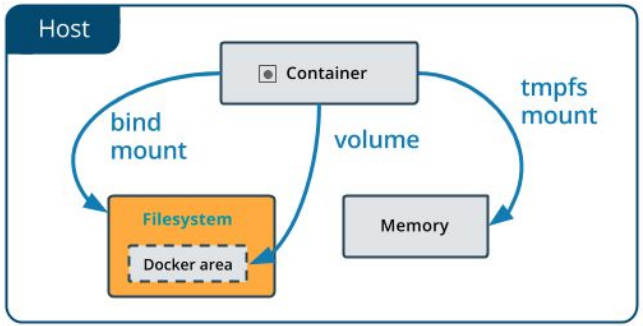
Por defecto, todos los archivos creados dentro de un contenedor se almacenan en una capa de escritura del contenedor. Esto significa que:

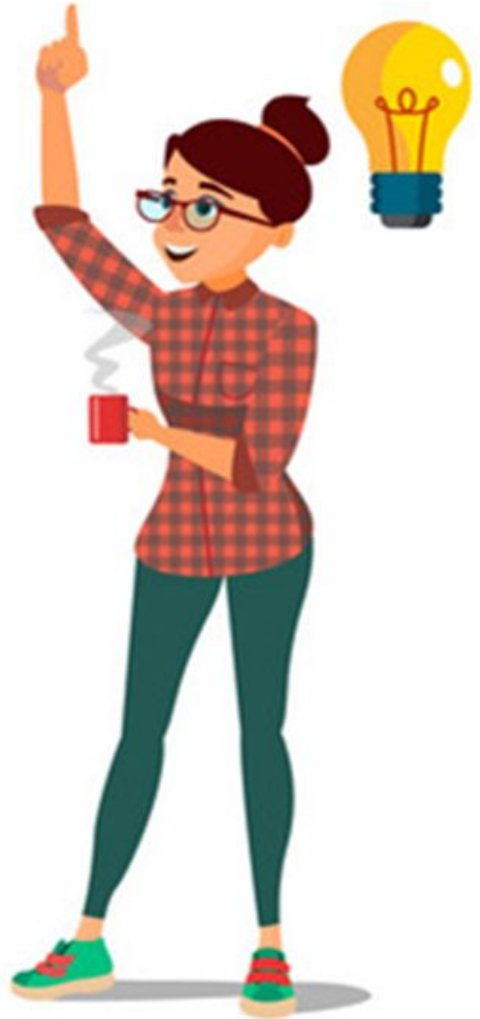
- Los datos no persisten cuando ese contenedor ya no existe y puede ser difícil sacar los datos del contenedor si otro proceso los necesita.
- La capa de escritura de un contenedor está estrechamente acoplada a la máquina host donde se ejecuta el contenedor. No puede mover fácilmente los datos a otro lugar.
- Escribir en la capa de escritura de un contenedor requiere un controlador de almacenamiento para administrar el sistema de archivos. Esta abstracción adicional reduce el rendimiento.

Volúmenes

Docker tiene varias opciones para que los contenedores almacenen archivos en la máquina host, de modo que los archivos persistan incluso después de que el contenedor se detenga:

- volúmenes (Archivo en un directorio)
- bind mount (Directorio del host)
- tmpfs mount (Temporal, muere con el contenedor)



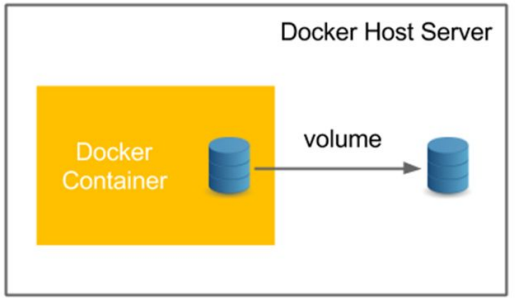


Volúmenes

Los volúmenes son el mecanismo preferido para conservar los datos generados y utilizados por los contenedores de Docker. Los volúmenes tienen varias ventajas:

- Son fáciles de respaldar o migrar.
- Se pueden administrar mediante los comandos CLI de Docker.
- Funcionan en contenedores de Linux y Windows.
- Se pueden compartir de forma más segura entre varios contenedores.

Además, los volúmenes suelen ser una mejor opción que persistir datos en la capa de escritura de un contenedor, porque un volumen no aumenta el tamaño de los contenedores que lo utilizan y el contenido del volumen existe fuera del ciclo de vida de un contenedor determinado.



Volúmenes

Tipos de volúmenes

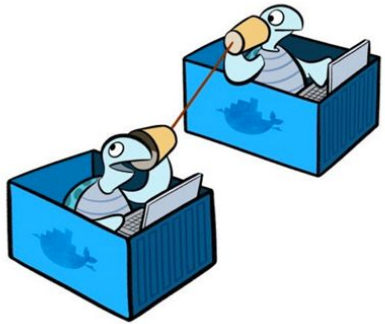
- **Volúmenes de host**: se puede acceder desde un contenedor de Docker y se almacena en el host en una ubicación definida por nosotros. Se sugiere utilizar un volumen de host cuando necesite saber dónde consultar los datos. También es el tipo de volumen más fácil de usar, por lo que es ideal para proyectos simples.

- **Volúmenes anónimos**: Docker administra la su ubicación. Puede ser difícil hacer referencia al mismo volumen cuando es anónimo. Estos volúmenes brindan flexibilidad, pero no se usan con tanta frecuencia ahora que se han introducido los volúmenes nombrados.

- **Volúmenes nombrados**: Docker administra dónde están ubicados. Sin embargo, se puede hacer referencia a los volúmenes con nombre por nombres específicos. Al igual que los volúmenes anónimos, los volúmenes con nombre brindan flexibilidad, pero también son explícitos. Esto los hace más fáciles de manejar.

Network

Network



Docker incluye soporte de red mediante el uso de controladores de red.

También se pueden escribir plugins de red para crear tus propios controladores, pero esa es una tarea avanzada.

Cada instalación de Docker Engine incluye automáticamente tres redes predeterminadas:

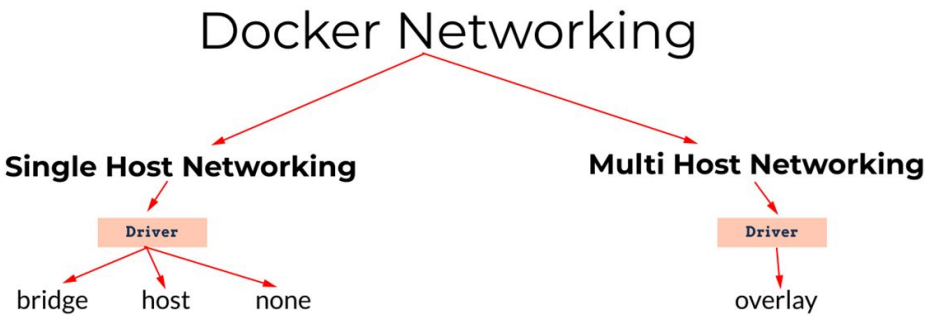
```
$ docker network ls

NETWORK ID          NAME                DRIVER
18a2866682b8        none                null
c288470c46f6        host                host
7b369448dccb        bridge              bridge
```

Network

Tipos de redes

- **Bridge**: el driver de red predeterminado. Si no se especifica un controlador, este es el tipo de red que está creando.
- **Host**: elimina el aislamiento de la red entre el contenedor y el host de Docker y usa la red del host directamente.
- **Overlay**: conecta varios Docker host y habilita la comunicación entre ellos.
- **None**: deshabilita todas las redes. Por lo general, se usa junto con un controlador de red personalizado.
- **Plugin de red**: se puede instalar y usar complementos de red de terceros con Docker.



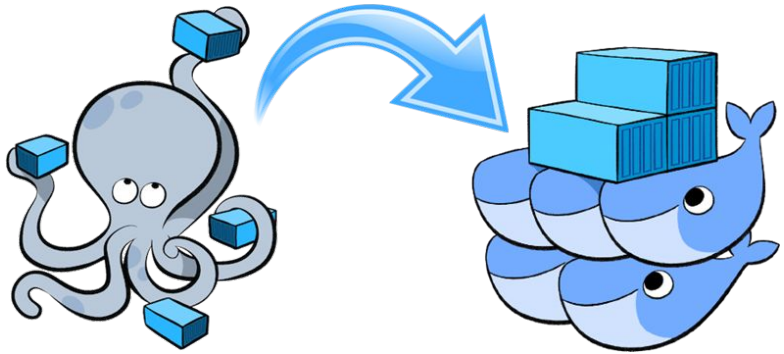
Docker compose

Docker Compose

Docker Compose es una herramienta para definir y ejecutar aplicaciones Docker de varios contenedores.

Con Compose, utiliza un archivo YAML para configurar los servicios de la aplicación. Luego, con un solo comando, crea e inicia todos los servicios desde la configuración.

Para instalar componer:
https://docs.docker.com/compose/install/



Docker Compose

El uso de Compose es básicamente un proceso de tres pasos:

1. Definir el entorno de la aplicación con un Dockerfile.
2. Definir los servicios que componen la aplicación en docker-compose.yml para que puedan ejecutarse juntos.
3. Ejecutar “docker compose up”. Así, se inicia y ejecuta toda la aplicación.

Docker Compose

Un **docker-compose.yml** se ve así:

```yaml
version: "3.9"  # optional since v1.27.0
services:
  web:
    build: .
    ports:
      - "5000:5000"
    volumes:
      - .:/code
      - logvolume01:/var/log
    links:
      - redis
  redis:
    image: redis
volumes:
  logvolume01: {}
```

version (obligatorio)
services(obligatorio)
volumes (opcional)
networks(opcional)

Kubernetes

Kubernetes

En sistemas a gran escala, las aplicaciones contenerizadas se vuelven difíciles de gestionar manualmente porque suelen incluir cientos e incluso miles de contenedores.

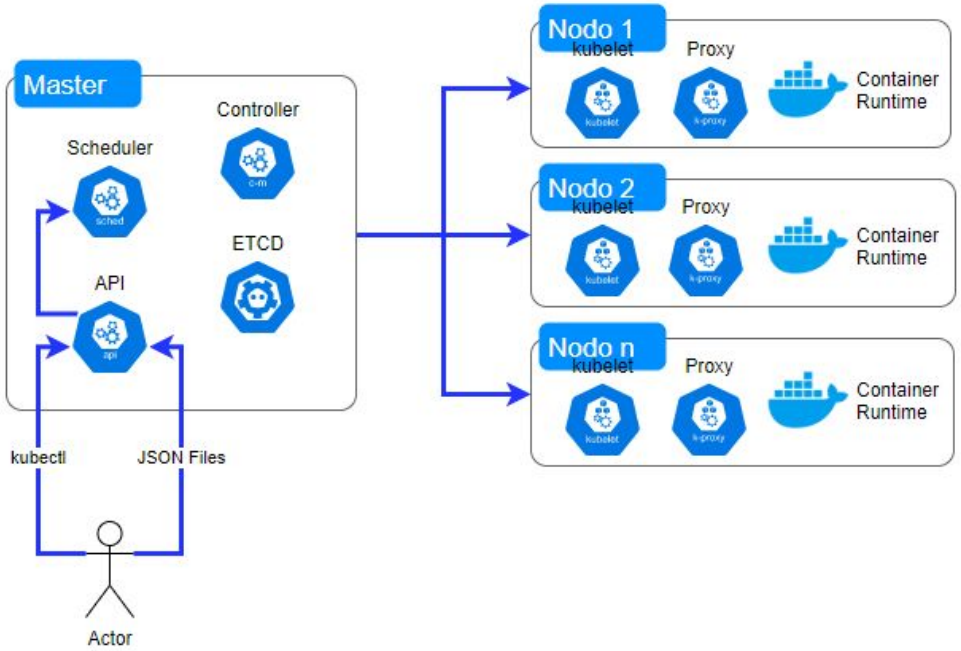
La orquestación de contenedores es esencial para reducir la complejidad operacional a la hora de ejecutarlos.

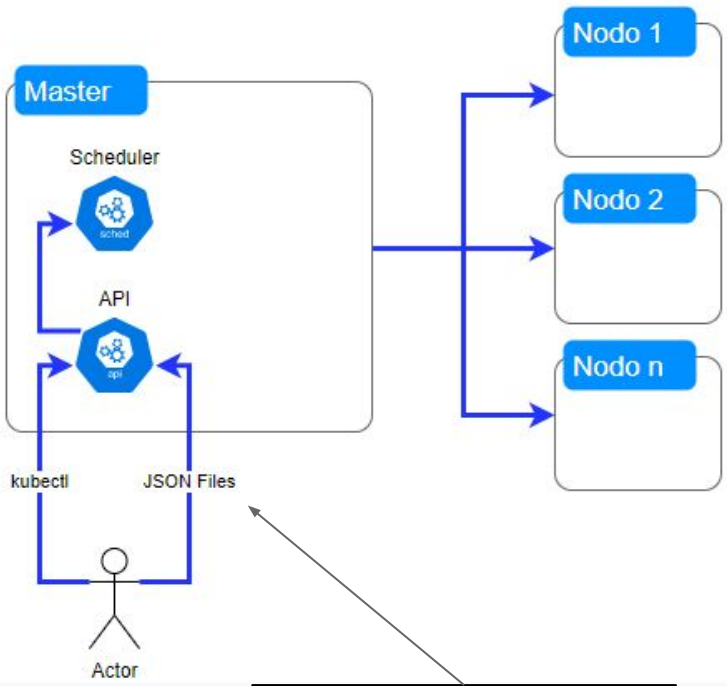
Kubernetes (K8s) es una plataforma para la orquestación de contenedores de código abierto que automatiza la implementación, el escalado y la administración de aplicaciones en contenedores.

kubernetes

A groso modo, los componentes de k8s se dividen en:

- **master**: donde viven los componentes de k8s, el cerebro de k8s, el que da las órdenes, se comunica con los nodos y les dice qué hacer, es el centro de operaciones. Puede haber varios masters para garantizar alta disponibilidad.
- **nodos**: máquinas virtuales o físicas donde podemos correr contenedores.

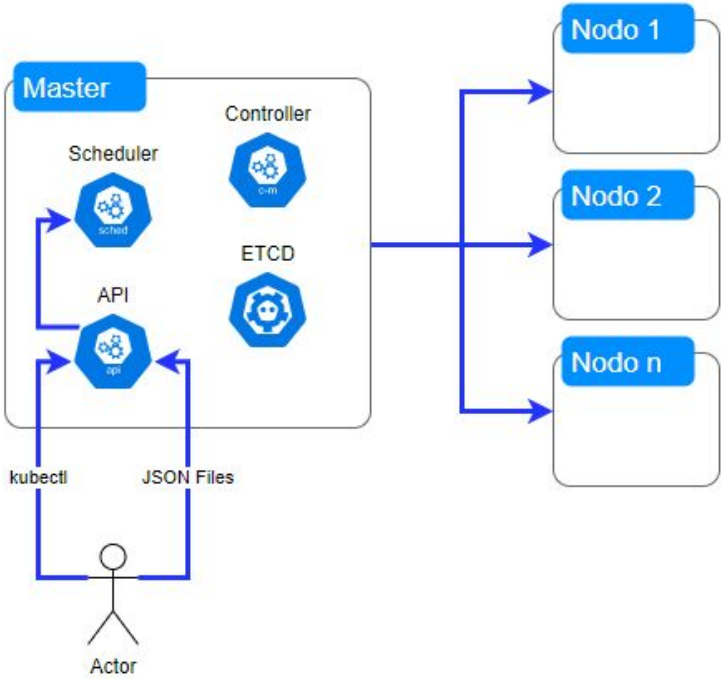




Un **manifiesto** es un archivo en YAML que define el recurso que queremos usar en Kubernetes. Facilita la administración. Para obtener el manifiesto

kube-apiserver: Nos comunicaremos con k8s a través de su API. La API acepta llamados en formato request en JSON. También, usaremos la línea de comando “kubectl” para comunicarnos desde nuestra terminal hacia k8s a través de la API.

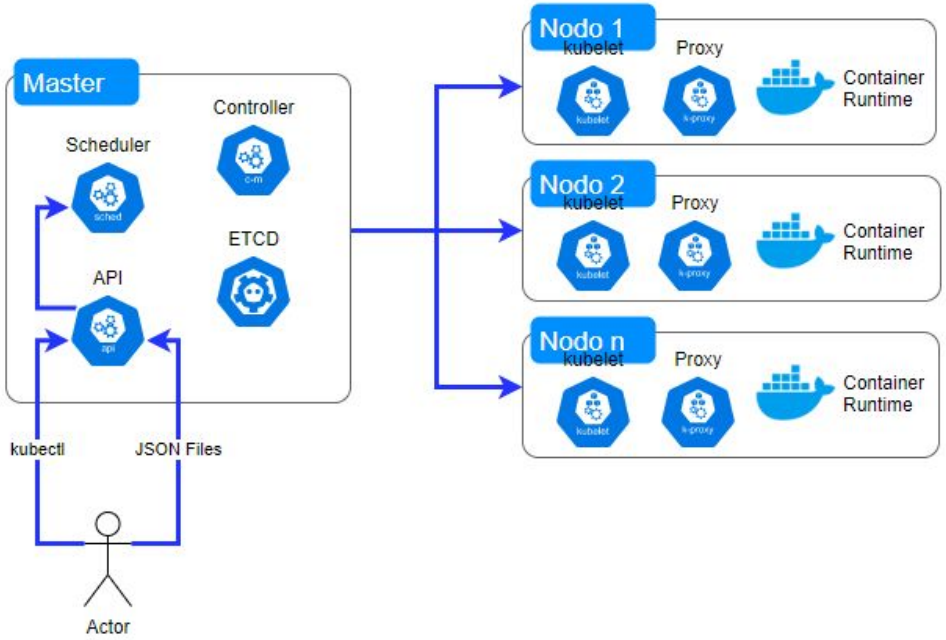
kube-scheduler: Cuando se hace una llamada a la API, la API se comunica con el Scheduler para ejecutar la acción. Cuando hay muchos nodos (+3000), el Scheduler tomará los primeros 15 y los evaluará respecto a su rendimiento para ver en dónde lanzar el pod (contenedor), en el caso de que la acción sea levantar uno nuevo.
En el caso de que el Scheduler no encuentre un nodo que sirva para levantar el pod, dejará la solicitud en estado pendiente.



kube-controller: El controlador contiene:

- Node-Controller: el controlador de los nodos (si se cae uno, levanta otro)
- Replication-Controller: encargado de mantener réplicas en el cluster
- Endpoint-Controller: controla servicios y pods a nivel de redes
- ServiceAccountAndToken-Controller: controla temas de autenticación de llamadas a la API

ETCD: es una base de datos de llave-valor en donde el cluster almacena el estado, los datos, configuraciones, backups, etc.



kubelet: Para que el master se pueda comunicar con cada nodo, el nodo tiene que tener instalado un servicio kubelet. Tiene dos funciones:

- recibir órdenes desde el master y enviar información al master
- comunicarse con el runtime de contenedores (ej: docker) del host

kube-proxy: corre en cada nodo, es un servicio que se encarga de las redes de los contenedores en los pods.

Container Runtime: Cada nodo tendrá instalado un container runtime que será el encargado de correr los contenedores en cada máquina. Docker, es un ejemplo de container runtime.

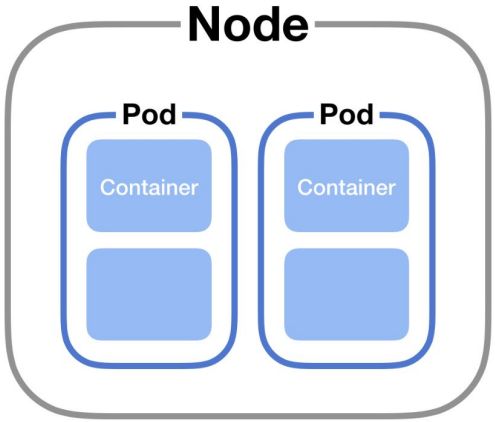
Los **Pods** son las unidades de computación desplegables más pequeñas que se pueden crear y gestionar en Kubernetes.

Un **Pod** es un grupo de uno o más contenedores con almacenamiento y red compartidos, y unas especificaciones de cómo ejecutar los contenedores.

Los contenidos de un Pod son siempre coubicados, coprogramados y ejecutados en un contexto compartido.

Al igual que los contenedores de aplicaciones individuales, los Pods se consideran entidades relativamente efímeras (en lugar de duraderas).

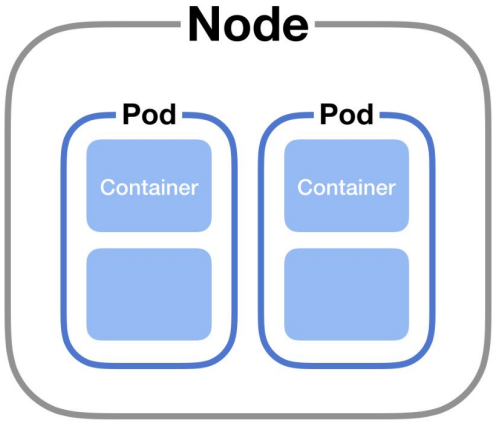
El **Pod**, en si mismo, no “corre”, sólo es un recubrimiento para permitir la comunicación entre los contenedores.



Problemas de los Pods

1. No se recuperan solos (si uno se muere, no se regenera)
2. Por sí mismo, no pueden replicarse (si necesitan 50 copias de sí mismo, no pueden hacerlo)
3. Los pods no pueden actualizarse a sí mismo (no puedo actualizar, por ejemplo, un comando que se ejecuta)

Para solucionar estos problemas se usan objetos de alto nivel.

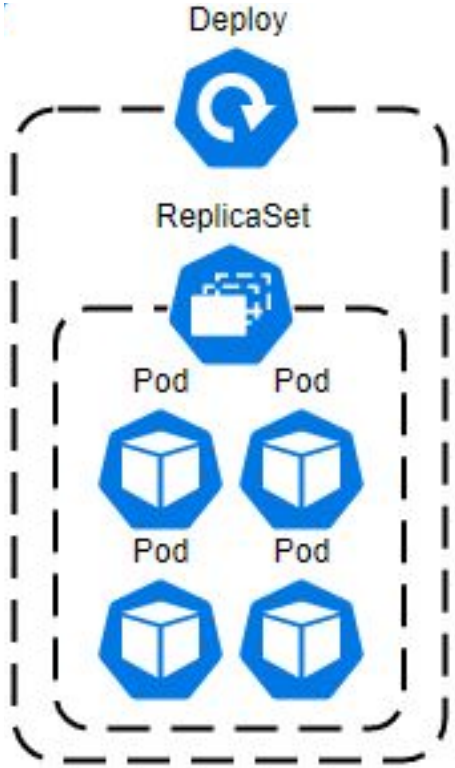


ReplicaSets

Es un objeto separado del pod y que está a un nivel más alto. Crea pods a partir de un template. Se define un número de réplicas deseadas. Mantiene el número de pods deseados, es decir, si un pod muere, el ReplicaSet lo genera solo.

Problemas de los ReplicaSets

Si ejecutamos cambios en el template del pod, no ocurre nada ya que el ReplicaSet solo se encarga de mantener un número de pods. Para poder manejar esto, existe un objeto de un nivel más alto que lo resuelve: los deployments.



Deployments

Al crear un Deployment, se define un template de ReplicaSet y se crea el propio ReplicaSet con un número de versión. Cuando actualizamos el deploy, Deployments crea un nuevo ReplicaSet (versión 2). Este proceso garantiza que los pods siempre estén vivos.

¡GRACIAS!


Las 9 tendencias tecnológicas que transformarán los servicios públicos y mejorarán la vida de los ciudadanos
Blockchain
• Blockchain and Distributed Ledgers
Inteligencia Artificial
• Cognitive Expert Advisors
• Conversational Systems
• Deep Learning
• Natural Language Processing
Citizen Engagement Analytics
• Customer Engagement Hub
• Personal Analytics
• Social Media Analytics
Tecnologías Cloud y Supercomputación
• Cloud Service / Data Brokerage (Open Data)
• FOG Cloud. Global Dynamic Automation from Servers to Mobile Devices
• High Performance Computing
• Native Cloud Frameworks for Mesh App and Microservices
• PaaS and Hybrid Cloud Enabling Platforms
TECNOLOGÍAS
Nanotecnología
• Nano - Machines
• Photonics
• Quantum Technologies
Nuevos Procesos de Producción
• 3D Printing
Robótica y Espacio
• Autonomous Vehicles
• Cubesats
• Robotic Process Automation
• Social Robots
• Unmanned Aerial Vehicles
Mundo Virtual
• Digital Twins
• Virtual and Augmented Reality
Plataformas Inteligentes
• Big Data Platforms
• Communication Network
• IoT Platforms
Fuente: Indra