Cloud Computing para Inteligencia Artificial

Sesión 4: Servicios de Computación II - Computación II - Contenedores y Serverless

Profesor: Dr. Martínez

Grado: Ingeniería en Inteligencia Artificial

Agenda de la Sesión

01

El Problema a Resolver

Del "Funciona en mi máquina" a la producción

02

Contenedores: Docker

Aislamiento y portabilidad

03

Orquestación a Escala

La necesidad de Kubernetes (K8s)

04

Servicios AWS

ECS, EKS y ECR

05

Paradigma Serverless

Funciones como Servicio con AWS Lambda

06

Aplicaciones en IA

Casos de uso prácticos

07

Comparativa Cloud

GCP y Azure

08

Resumen y Preguntas

Conclusiones y próximos pasos

¿Por qué necesitamos más que una VM?

Hemos visto las Máquinas Virtuales (EC2), pero ¿es siempre la mejor opción?

Problema Clásico: Un modelo de scikit-learn entrenado con Python 3.8 y Pandas y Pandas 1.5.1 funciona perfectamente en tu portátil. Al desplegarlo en una una instancia EC2 con Python 3.9 y Pandas 2.0, falla por incompatibilidades sutiles. incompatibilidades sutiles.

Desafíos:

- **Consistencia de Entornos:** Garantizar que desarrollo, pruebas y producción sean idénticos
- **Gestión de Dependencias:** Evitar el "infierno de las dependencias"

- **Eficiencia de Recursos:** Las VMs son pesadas y a menudo desperdician RAM y desperdician RAM y CPU
- **Portabilidad:** ¿Cómo movemos fácilmente cargas de trabajo entre nubes?

Virtualización vs. Contenerización

Máquinas Virtuales (VMs) (VMs)

- Virtualizan el **hardware**
- Cada VM tiene su propio Sistema Operativo completo (Guest OS), además del SO del host
- Pesadas: Ocupan **gigabytes** de espacio
- Lentas: Tardan **minutos** en arrancar
- Aislamiento Fuerte: El hipervisor proporciona una barrera de seguridad a nivel de hardware

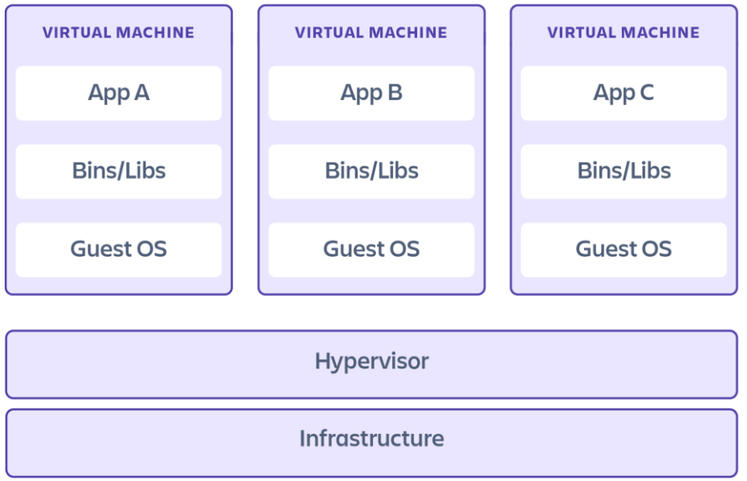
Contenedores

- Virtualizan el **Sistema Operativo**
- Comparten el kernel del SO del host. host. Contienen solo la aplicación y sus y sus dependencias
- Ligeros: Ocupan **megabytes**
- Rápidos: Arrancan en **segundos o milisegundos**
- Aislamiento Suficiente: A nivel de procesos, namespaces y cgroups en Linux

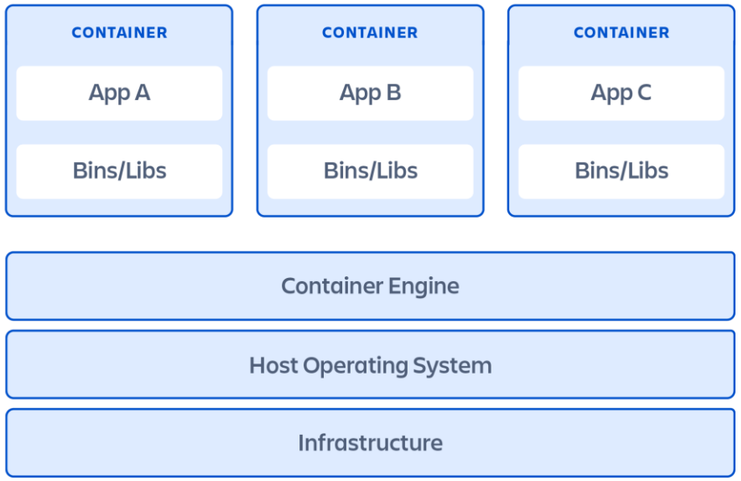
Virtualización vs. Contenerización

Contenerización

Virtual machines



Containers



Docker: El Estándar de Facto

Docker es una plataforma open-source para construir, distribuir y ejecutar contenedores.

Imagen Docker (Image)

Una plantilla inmutable y de solo lectura con instrucciones para crear un contenedor. Se construye por capas (Ubuntu → Python → dependencias → código).

Dockerfile

Un fichero de texto que define, paso a paso, cómo construir una imagen Docker. Es "infraestructura como código" para tu aplicación.

Contenedor Docker

Una instancia en ejecución de una imagen. Es el proceso aislado y ejecutable que contiene tu contiene tu aplicación.

Registro (Registry)

Un repositorio para almacenar y distribuir imágenes. Docker Hub (público), AWS ECR, GCP Artifact Registry y Azure CR (privados).

Anatomía de un Dockerfile para IA

Veamos un ejemplo para empaquetar una API de inferencia con Flask y TensorFlow.

```
# 1. Usar una imagen base oficial con Python
     FROM python:3.9-slim
# 2. Establecer el directorio de trabajo
     WORKDIR /app
# 3. Copiar el fichero de dependencias
     COPY requirements.txt .
# 4. Instalar las dependencias
     RUN pip install --no-cache-dir -r requirements.txt
# 5. Copiar el código de la aplicación
     COPY . .
# 6. Exponer el puerto de la aplicación
     EXPOSE 5000
# 7. Comando para ejecutar la aplicación
     CMD ["python", "app.py"]
```

Comandos básicos:

`docker build -t mi-api-ia .` (Construye la imagen)

`docker run -p 5000:5000 mi-api-ia` (Ejecuta el contenedor)

Beneficios de los Contenedores para IA/ML

Reproducibilidad de Experimentos

Un Dockerfile fija la versión del SO, de Python, de las librerías (tensorflow==2.11.0), las variables de entorno y el código. ¡Cualquier miembro del equipo puede reproducir tu entorno exacto con un docker run!

Encapsulación de Modelos

El modelo entrenado (.h5, .pkl) y todo su entorno de ejecución ejecución se empaquetan juntos. Se convierte en un artefacto artefacto portable y versionable, listo para desplegarse en en cualquier lugar.

Despliegue Consistente

El mismo contenedor que se probó en local es el que se despliega en la nube para servir predicciones (inferencia). Elimina sorpresas en producción y reduce errores.

Aislamiento de Dependencias

Proyectos de IA a menudo tienen dependencias conflictivas (ej: conflictivas (ej: diferentes versiones de CUDA). Los contenedores aíslan estos entornos perfectamente.

El Desafío de la Escala

Ejecutar un contenedor es fácil. Pero, ¿qué pasa cuando tenemos una aplicación de IA con múltiples microservicios y miles de usuarios?

Preguntas a resolver:

- **Despliegue y Escalado:** ¿Cómo lanzo 10 réplicas de mi contenedor de inferencia? ¿Y si necesito 50?
- **Alta Disponibilidad:** ¿Qué pasa si el servidor donde corre mi corre mi contenedor se cae?
- **Balanceo de Carga:** ¿Cómo distribuyo las peticiones entre todas las réplicas?
- **Descubrimiento de Servicios:** ¿Cómo el "microservicio A" A" encuentra al "microservicio B" si sus IPs son dinámicas? dinámicas?
- **Actualizaciones sin Caídas:** ¿Cómo actualizo mi modelo sin interrumpir el servicio?

Hacer esto manualmente es imposible. **Necesitamos un orquestador.**

Kubernetes (K8s): El Sistema Operativo del Cloud

Kubernetes (del griego κυβερνήτης, "timonel") es una plataforma plataforma open-source que automatiza el despliegue, escalado y escalado y gestión de aplicaciones en contenedores.

Es el orquestador líder del mercado y el estándar de facto.

Ventajas:

- **Portabilidad:** Funciona en cualquier proveedor cloud (AWS, GCP, Azure) y on-premise. Evita el vendor lock-in
- **Gran Ecosistema:** Enorme comunidad y miles de herramientas que se integran con él (Prometheus, Helm, etc.)
- **Auto-reparación:** Si un contenedor falla, K8s lo reinicia. Si un nodo muere, K8s mueve sus contenedores a nodos sanos

Kubernetes ofrece una capa de abstracción sobre la infraestructura, permitiendo a los desarrolladores de IA centrarse en sus modelos y aplicaciones en lugar de en los detalles de los servidores.

Conceptos Clave de Kubernetes

Vista a 10.000 pies de los componentes fundamentales:

Cluster

El conjunto de todas las máquinas máquinas (nodos) que gestiona Kubernetes.

Nodo (Node)

Una máquina, física o virtual (ej: una instancia EC2).

- **Control Plane:** El cerebro del cluster. Toma decisiones globales
- **Worker Node:** La máquina que ejecuta los contenedores

Pod

La unidad de despliegue más pequeña en K8s. Wrapper alrededor de uno o más contenedores. Generalmente, 1 Pod = 1 contenedor. Los pods son efímeros.

Deployment

Declara el estado deseado. Ejemplo: "Quiero 3 réplicas de mi-api-ia v1.2". Se encarga de crear los Pods y mantenerlos.

Service

Proporciona IP y DNS estables para un conjunto de Pods. Permite la comunicación entre servicios, actuando como balanceador de carga interno.

¿Gestionar K8s o Usarlo? El Rol del Cloud

Instalar y mantener un cluster de Kubernetes es complejo. Requiere expertos en redes, seguridad y sistemas.

El dilema:

¿Queremos dedicar nuestro tiempo a ser expertos en Kubernetes o a construir modelos de IA?

Los proveedores de nube ofrecen **Servicios de Contenedores Gestionados**: Ellos se encargan de la complejidad del orquestador (el "Control Plane"), y nosotros solo nos preocupamos por desplegar nuestras aplicaciones.

En AWS, tenemos dos caminos principales:

- El camino "propiedad de AWS": ECS
- El camino "estándar open-source": EKS

Amazon ECS (Elastic Container Service)

Es el orquestador de contenedores propietario de AWS.

Conceptos clave:

- **Task Definition:** Un blueprint (JSON) que describe qué imagen Docker usar, cuánta CPU/Memoria, puertos, etc. Similar a un docker-compose
- **Task:** Una instancia en ejecución de una Task Definition. Similar a un Pod en K8s
- **Service:** Mantiene un número deseado de Tasks en ejecución y se ejecución y se integra con balanceadores de carga (ALB)
- **Cluster:** Un agrupamiento lógico de recursos donde se ejecutan las Tasks

Ventaja Principal:

Mucho más simple de aprender y usar si ya estás familiarizado con el con el ecosistema de AWS (IAM, VPC, ELB). Integración nativa y y profunda con otros servicios AWS.

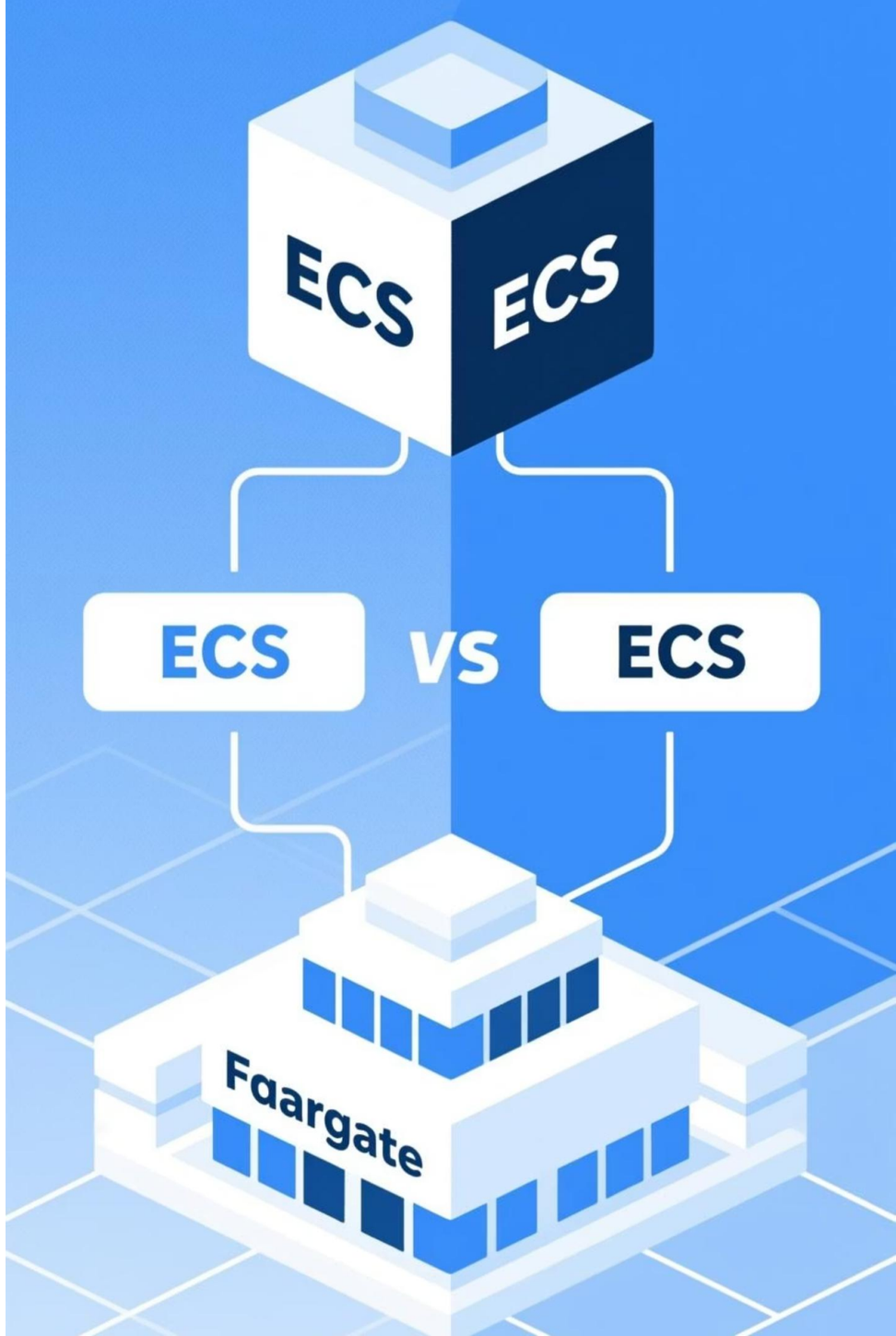
ECS: Modos de Lanzamiento

Modo EC2

- Tú provisionas y gestionas un cluster de instancias EC2
- Tú eres responsable de parchear, escalar y securizar el SO
- Control total: Puedes elegir el tipo de instancia (GPUs), usar EBS, etc.
- Pago: Pagas por las instancias EC2 24/7, independientemente de si ejecutan contenedores o no

Modo Fargate (Serverless)

- No gestionas servidores. AWS provisiona la infraestructura "just-in-time"
- Simplemente defines tu Task y Fargate la ejecuta
- Simplicidad máxima: Te olvidas del olvidas del sistema operativo y del y del escalado
- Pago: Solo por la vCPU y memoria que tus contenedores consumen mientras se ejecutan



Amazon EKS (Elastic Kubernetes Service)

Es el servicio de Kubernetes gestionado de AWS.

- AWS gestiona la disponibilidad y escalabilidad del **Control Plane** de Kubernetes
- Tú gestionas los **Worker Nodes** (instancias EC2) o puedes puedes usar Fargate para Pods serverless
- Certificado por la CNCF: Es un Kubernetes "puro". Aplicaciones K8s estándar correrán en EKS

¿Cuándo elegir EKS?

- Cuando necesitas la potencia y el ecosistema completo de Kubernetes
- Cuando quieres una estrategia multi-cloud o híbrida
- Cuando tu equipo ya tiene experiencia en K8s

Consideraciones de IA: GPUs y Kubeflow

Uso de GPUs

Tanto ECS (con EC2) como EKS permiten usar instancias con GPU (familias p y g de EC2). Crucial para inferencia de modelos de Deep Learning.

En K8s/EKS, se requiere el NVIDIA device plugin for Kubernetes para exponer las GPUs a los contenedores.

Kubeflow

Proyecto open-source que convierte a Kubernetes en una plataforma de MLOps.

- Pipelines de Machine Learning portables y escalables
- Jupyter Notebooks como servicio
- Servicio de modelos (KFServing/KServe) (KFServing/KServe)
- Hyperparameter Tuning

Se puede instalar sobre EKS para crear una potente plataforma de IA.

¡Elige la Arquitectura Correcta!

Escenario: Eres el arquitecto de una startup de IA. Debes desplegar un nuevo servicio de "traducción de texto en tiempo real" bas
Transformer.

Requisitos

1. Tráfico muy variable: picos intensos y horas de inactividad
2. Equipo pequeño sin expertos en Kubernetes
3. Despliegue rápido y simple
4. Coste mínimo durante horas de inactividad

¿Qué servicio de AWS elegirías para desplegar el contenedor de la API?

A) Amazon EKS con nodos EC2 con GPU

Potente pero complejo y caro si está inactivo

B) Amazon ECS con Fargate

Simple, serverless, paga por uso, ideal para tráfico variable

C) Una única instancia EC2 con Docker

Poco escalable y sin alta disponibilidad

Más Allá de los Contenedores: Serverless

Hemos visto "serverless para contenedores" con Fargate. Pero hay un Pero hay un nivel de abstracción aún mayor: **Funciones como Servicio Servicio (FaaS)**.

Principios Clave del Serverless:

1. **Sin Gestión de Servidores:** No hay que provisionar, parchear ni escalar servidores. Ni siquiera contenedores o clusters
2. **Escalado Automático e Instantáneo:** La plataforma escala de cero de cero a miles de ejecuciones en paralelo según la demanda demanda
3. **Pago por Ejecución:** Si tu código no se ejecuta, no pagas nada. Pagas por milisegundo de ejecución y memoria asignada

Es la computación más orientada a eventos. Tu código "duerme" hasta "duerme" hasta que algo (un evento) lo despierta.

El paradigma serverless lleva la abstracción al extremo: el desarrollador se centra únicamente en la

AWS Lambda: Tu Código en la Nube

Lambda es el servicio de FaaS de AWS. Te permite ejecutar código sin pensar en servidores.

Concepto Central:

Escribes una función en tu lenguaje preferido (Python, Node.js, etc.) y la subes a Lambda.

Triggers (Disparadores)

Petición HTTP

A través de API Gateway. Crear un endpoint de API REST

Subida a S3

Un nuevo objeto en un bucket activa la función

Mensaje en SQS

Procesamiento de mensajes en cola

DynamoDB

Cambio en una tabla de base de datos

Programada

Cada hora, cada día (con EventBridge)

¡Y muchos más!

Más de 200 integraciones en AWS

Modelo de Ejecución de Lambda

Handler

El punto de entrada de tu código. Es la función que Lambda ejecutará.

```
def handler(event, context):    # event: datos del evento
que disparó la función    # context: información sobre la
invocación    return {"statusCode": 200, "body": "Hello
World"}
```

Entorno de Ejecución

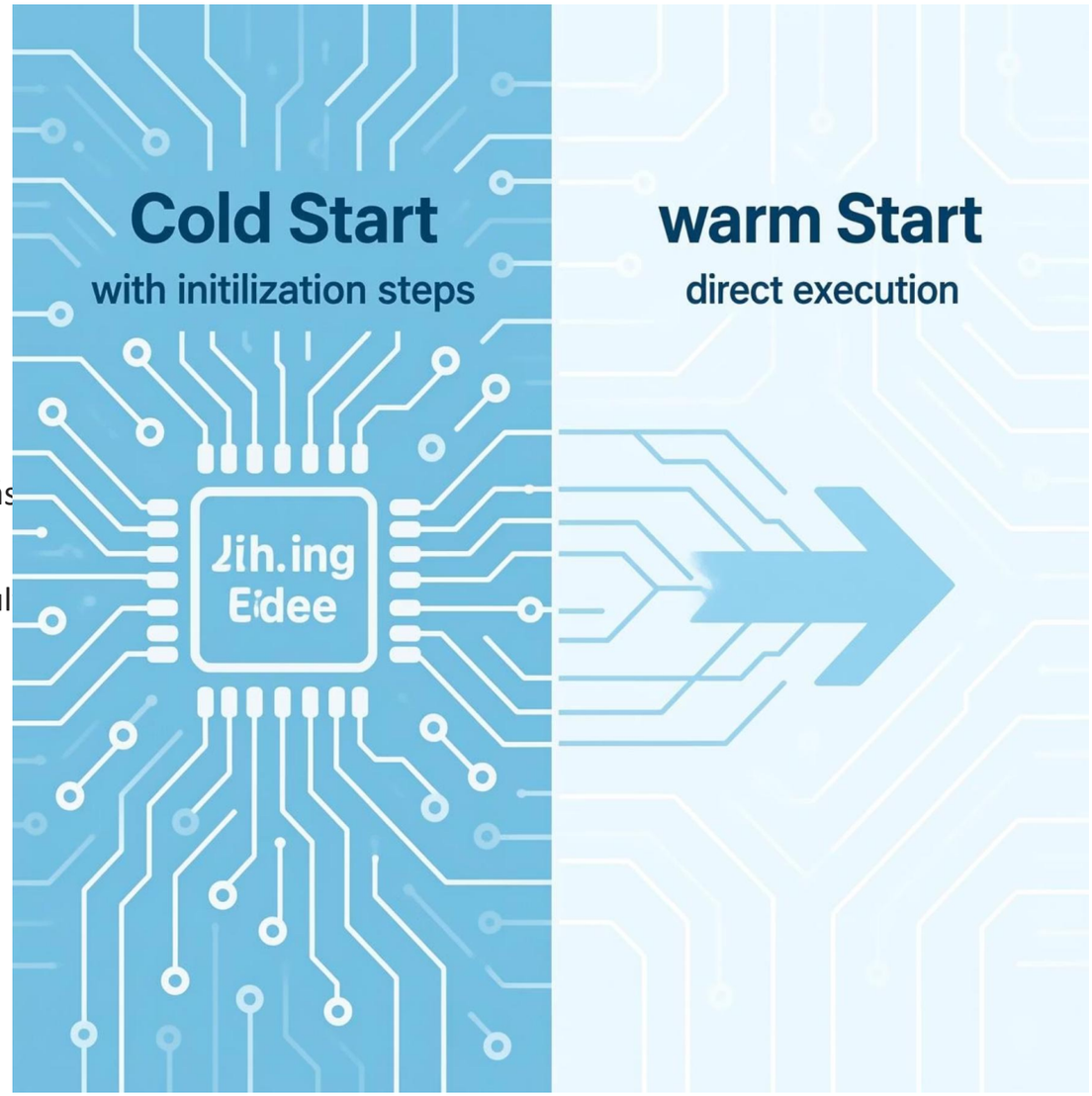
Un entorno temporal y aislado con tu código, el runtime del lenguaje y las dependencias.

Cold Start vs Warm Start

- **Cold Start:** Primera invocación o tras inactividad. Lambda crea el entorno desde cero: descarga código, inicia runtime... Añade latencia (ms a segundos)
- **Warm Start:** Invocación rápida tras uso reciente. Lambda reutiliza el entorno. Mucho más

Layers

Forma de empaquetar librerías y dependencias (NumPy, Pandas) para compartirlas entre múl



Lambda para IA: Casos de Uso

Lambda tiene limitaciones (15 min de ejecución, tamaño de paquete limitado). No es para entrenar modelos grandes, pero es perfecto para tareas cortas y basadas en eventos.

Preprocesamiento de Datos

Trigger: Una imagen se sube a un bucket S3

Lambda: Redimensiona la imagen, extrae metadatos, la normaliza y la guarda en otro bucket, lista para entrenamiento

Inferencia en Tiempo Real

Trigger: Petición POST a un endpoint de API Gateway

Lambda: Carga un modelo ligero (regresión logística, árbol de decisión, modelo NLP pequeño), realiza la predicción y devuelve el resultado

Orquestación de Flujos de ML

Herramienta: AWS Step Functions puede orquestar un pipeline complejo llamando a diferentes Lambdas en secuencia o paralelo

Ejemplo: Una función para validar datos, otra para invocar entrenamiento, otra para desplegar

Serverless para Contenedores: Lo Mejor de Dos Mundos

A veces una Lambda no es suficiente (dependencias binarias específicas, runtime personalizado, paquete grande), pero aún quieres la simplicidad de no gestionar servidores.

Solución: Ejecutar tus contenedores en una plataforma serverless

AWS Fargate

Como vimos, ejecuta contenedores de ECS o EKS sin gestionar nodos

Google Cloud Run

Extremadamente popular. Despliegas un contenedor y GCP te da un endpoint HTTPS. Escala automáticamente, incluso a cero (no hay tráfico, no pagas)

Azure Container Instances

Permite ejecutar un único contenedor de forma forma rápida y sencilla. Bloque de construcción construcción serverless básico para contenedores en contenedores en Azure

Un Vistazo a GCP

Google Kubernetes Engine (GKE)

Considerado por muchos el servicio de K8s gestionado más maduro y maduro y avanzado. Su modo Autopilot gestiona no solo el Control Plane, Control Plane, sino también los nodos, el escalado y la seguridad.

Cloud Run

Competidor directo para Fargate/ACI. Muy enfocado en la experiencia del experiencia del desarrollador.

Artifact Registry

Sucesor de Google Container Registry. Almacena imágenes de Docker y otros Docker y otros artefactos.

Cloud Functions

El FaaS de GCP, equivalente a AWS Lambda.

Un Vistazo a Azure

Azure Kubernetes Service (AKS)

El servicio gestionado de Kubernetes de Azure. Robusto y muy bien
bien integrado con el ecosistema de Azure (Azure Active Directory).
Directory).

Azure Container Instances (ACI)

La forma más rápida de ejecutar un contenedor en Azure sin orquestación.
orquestación.

Azure Container Registry (ACR)

Registro de contenedores privado y gestionado.

Azure Functions

El FaaS de Azure. Destaca por sus "Durable Functions" para orquestar flujos
orquestar flujos de trabajo con estado.

Contenedores vs. Serverless (FaaS): ¿Cuándo Usar Qué?

No es una competición, son herramientas diferentes para problemas diferentes.

Usa Contenedores cuando:

- Tienes una aplicación existente que quieres migrar a la nube (lift-and-shift)
- Necesitas control total sobre el entorno de ejecución y el networking
- Tu aplicación tiene un trabajo de larga duración o mantiene conexiones persistentes
- Quieres usar el ecosistema de Kubernetes (Kubeflow, Istio, etc.) Istio, etc.)
- La latencia de un cold start es inaceptable

Usa Serverless/FaaS cuando:

- Tu aplicación es orientada a eventos y de corta duración
- El tráfico es impredecible o esporádico
- La prioridad es la velocidad de desarrollo y el coste cero en inactividad
- Estás construyendo microservicios muy pequeños y desacoplados
- No quieres preocuparte por ningún aspecto de la infraestructura

Consideraciones Adicionales

Gestión del Estado

Tanto los contenedores (en orquestadores) orquestadores) como las funciones Lambda Lambda están diseñados para ser stateless stateless (sin estado). El estado de la aplicación debe externalizarse a servicios servicios como:

- Base de datos (RDS, DynamoDB)
- Caché (ElastiCache)
- Almacenamiento de objetos (S3)

Monitorización y Logging

¡Crítico! En estos sistemas distribuidos, necesitas una solución centralizada:

- **AWS CloudWatch:** Recolecta logs y y métricas de ECS, EKS y Lambda de de forma nativa
- **AWS X-Ray:** Para trazar peticiones a a través de múltiples servicios
- **En K8s:** Prometheus para métricas y Grafana para visualización

Depuración

La depuración en local es clave (usando Docker). La depuración en la nube es más compleja y se basa en el análisis de logs y trazas.

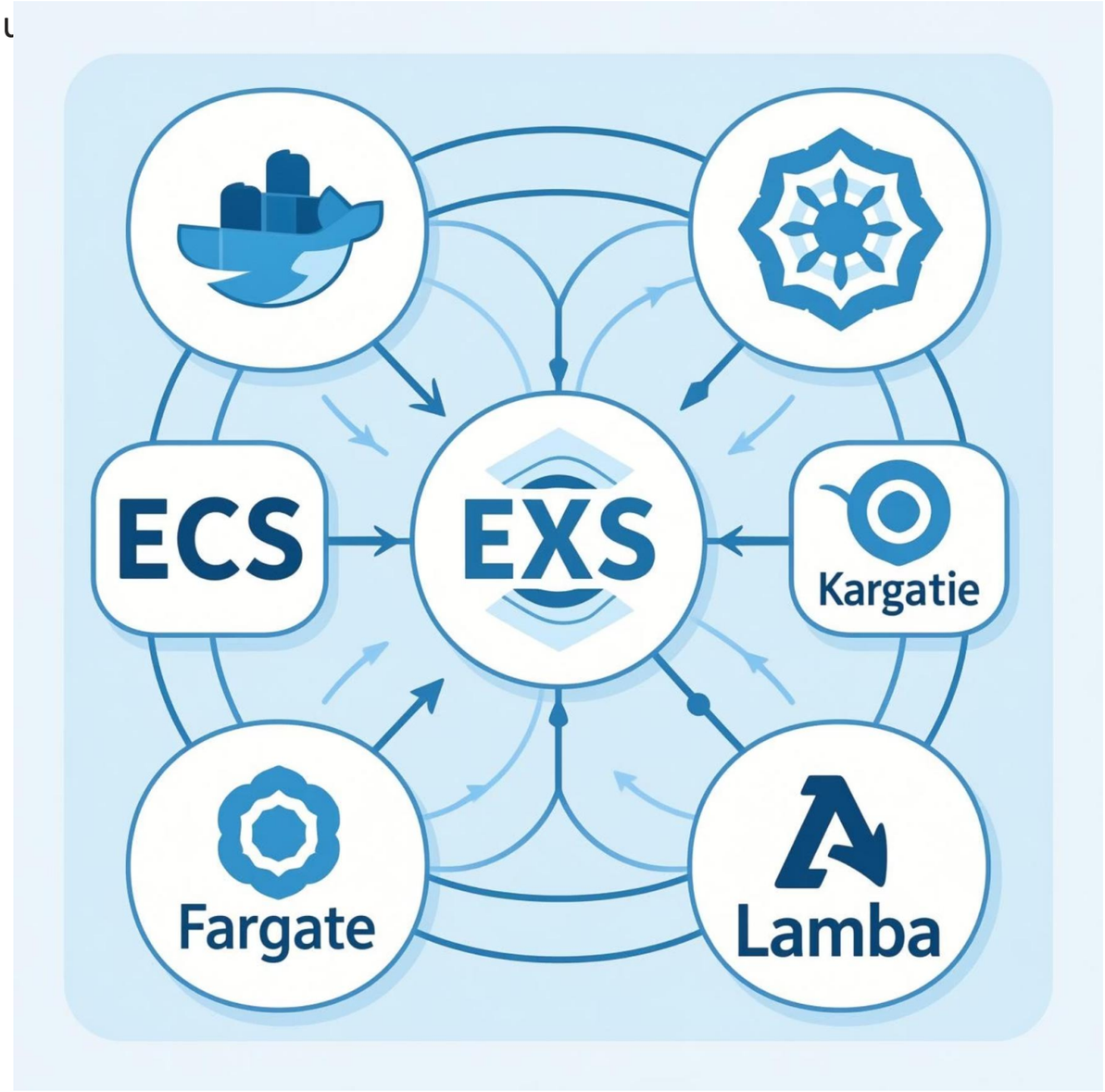
Considera herramientas como:

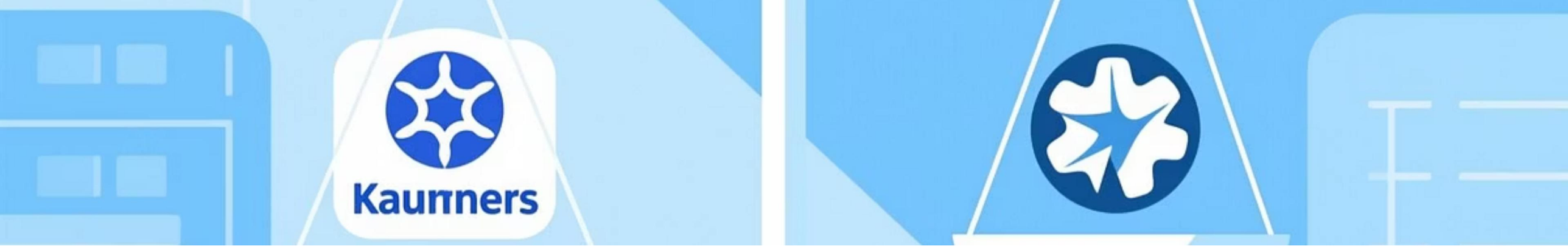
- AWS SAM para probar Lambdas localmente
- Docker Compose para entornos multi-contenedor
- Minikube para Kubernetes local

Resumen de la Sesión

Hemos cubierto conceptos fundamentales para el despliegu
de aplicaciones de IA:

- Los **contenedores (Docker)** resuelven el problema de la consistencia y portabilidad empaquetando la aplicación con sus dependencias
- **Kubernetes** se ha convertido en el estándar para orquestar contenedores a escala, pero es complejo de gestionar
- Servicios como **ECS y EKS** en AWS nos permiten usar orquestadores sin gestionar toda su complejidad
- **Fargate y Cloud Run** llevan los contenedores al mundo serverless, eliminando la gestión de servidores
- **Serverless FaaS (AWS Lambda)** es la máxima abstracción, ideal para código que se ejecuta en respuesta a eventos, pagando solo por el uso real
- La elección entre contenedores y serverless depende de la aplicación, el equipo y los requisitos de escalabilidad y control
- Para aplicaciones de IA completas, a menudo se utilizan combinaciones de combinaciones de estos servicios (enfoque híbrido)





Debate Rápido

Para servir un modelo de GPT-3.5-Turbo a través de una API que podría recibir millones de peticiones:

¿Qué arquitectura base te inspira más confianza y por qué?

Opción A: Kubernetes (EKS)

Argumentos a favor:

- Máximo control y granularidad
- Escalabilidad probada en entornos de producción masivos
- Uso de hardware específico (GPUs)
- Ecosistema maduro de herramientas para observabilidad
- Gestión avanzada de tráfico y políticas de red

Opción B: Serverless (Lambda + API Gateway)

Argumentos a favor:

- Escalabilidad masiva teóricamente infinita
- Gestión cero de infraestructura
- Coste potencialmente menor si el tráfico es irregular
- Tiempo de despliegue y puesta en marcha mínimo
- Arquitectura nativa de eventos

Referencias y Material Adicional

Documentación Oficial

- AWS ECS
- AWS EKS
- AWS Lambda
- Docker
- Kubernetes

Whitepapers

- "Running Containers on AWS" - AWS
- "The Twelve-Factor App" - Principios para aplicaciones cloud-native

Videos Recomendados

- "Serverless vs. Containers - Which is best for you?" - Fireship (YouTube)
- "AWS re:Invent 2022 - A closer look at AWS Fargate" - AWS Events

Cursos Online

- Docker & Kubernetes: The Practical Guide - Udemy/Coursera
- AWS Certified Developer Associate - A Cloud Guru
- Machine Learning Engineering for Production - Coursera

¿Preguntas?

Gracias por vuestra atención

Contacto

Email: profesor@universidad.es

Horario tutorías: Martes y Jueves 15:00-15:00-17:00

Despacho: Edificio de Informática, 3ª planta, 3.15

Recursos adicionales

Todos los materiales de la asignatura están disponibles en el campus virtual

Recordad reservar los equipos para el laboratorio práctico de la próxima semana semana

Las dudas también se pueden plantear en plantear en el foro de la asignatura