TÍTULO DEL PROYECTO

DISEÑO Y DESPLIEGUE DE UNA INFRAESTRUCTURA CLOUD DE ALTA DISPONIBILIDAD PARA PLATAFORMAS E-LEARNING (MOODLE) EN AWS MEDIANTE INFRAESTRUCTURA COMO CÓDIGO (TERRAFORM)

Autor: David Arbelaez Mutis

Tutor: [Nombre del Tutor]

Ciclo Formativo: Grado Superior en Administración de Sistemas Informáticos en Red (ASIR)

Centro Educativo: Prometeo (ThePower Education) - Campus Caja Mágica

Fecha: Enero 2026

RESUMEN EJECUTIVO (ABSTRACT)

El presente proyecto aborda la modernización de la infraestructura tecnológica necesaria para alojar una plataforma de aprendizaje en línea (LMS) Moodle. Partiendo de una solución monolítica inicial, se ha evolucionado hacia una arquitectura nativa en la nube (Cloud Native) en Amazon Web Services (AWS), alojada específicamente en la región de España (eu-south-2) para garantizar la soberanía del dato y la mínima latencia.

El desarrollo técnico se ha realizado siguiendo el paradigma de Infraestructura como Código (IaC) utilizando Terraform. La solución final implementa una arquitectura de tres capas con Alta Disponibilidad (HA), desacoplando la capa de computación (Auto Scaling Groups), la capa de datos (Amazon RDS) y el almacenamiento de archivos (Amazon EFS). Asimismo, se incluye un estudio de costes (FinOps) que justifica la migración desde modelos basados en IPs públicas fijas hacia modelos balanceados, demostrando una optimización financiera y operativa.

Palabras clave: AWS, Terraform, Moodle, Alta Disponibilidad, FinOps, Auto-Scaling, EFS, RDS.

ÍNDICE DE CONTENIDOS

1. INTRODUCCIÓN

1.1. Contexto del E-Learning

En el panorama educativo actual, la disponibilidad de las plataformas LMS (*Learning Management Systems*) es crítica. Una caída del servicio durante un periodo de exámenes o entregas puede tener consecuencias académicas graves. Las infraestructuras tradicionales basadas en servidores únicos (VPS) presentan un "Punto Único de Fallo" (SPOF) y carecen de la elasticidad necesaria para absorber picos de demanda.

1.2. Antecedentes y Justificación (Evolución del Proyecto)

Este proyecto no es un ejercicio teórico, sino el resultado de un proceso de ingeniería iterativo realizado a lo largo de 2025.

Fase 1: El Prototipo Monolítico (Bitnami)

Inicialmente, se desplegó una solución basada en una imagen "Bitnami Moodle" en una única instancia EC2. Aunque funcional para pruebas de concepto, este modelo evidenció graves carencias:

1. **Rigidez:** La base de datos y la aplicación residían en el mismo servidor.
2. **Ineficiencia Económica:** Se detectó un sobrecoste mensual debido a la política de

precios de AWS sobre las direcciones IPv4 públicas (Elastic IPs), necesarias para mantener el acceso al servidor único.
3. **Riesgo:** La actualización o fallo del servidor implicaba una parada total del servicio.

Detalles Técnicos del Hardware (Prototipo):

Para esta fase inicial se utilizó una instancia de la familia t3.micro con las siguientes especificaciones:

- **vCPU:** 2 vCPUs (Burstable Performance)
- **Memoria RAM:** 1 GiB
- **Almacenamiento:** Volumen EBS gp3 de 10 GB

Este dimensionamiento se basó en pruebas iniciales de bajo tráfico. Sin embargo, el análisis de rendimiento bajo carga reveló limitaciones. Según la documentación oficial de Moodle y métricas de la comunidad, el consumo de memoria RAM por proceso de usuario concurrente oscila entre **50MB y 100MB**. Esto implica que una instancia de 1 GB de RAM (como la t3.micro), tras descontar la memoria del sistema operativo y la base de datos local, solo puede soportar de forma estable entre **10 y 20 usuarios concurrentes** simultáneos.

Esta limitación física del hardware fue un factor determinante para migrar hacia una arquitectura escalable horizontalmente (Auto-Scaling). Si bien el escalado vertical (aumentar RAM/CPU de una sola máquina) podría resolver temporalmente el problema de rendimiento, mantendría el problema crítico de disponibilidad: seguiría siendo un Punto Único de Fallo (SPOF). Por el contrario, el escalado horizontal mediante un Auto Scaling Group no solo permite sumar capacidad agregando nodos, sino que garantiza la redundancia y la tolerancia a fallos, distribuyendo la carga y asegurando que el servicio continúe incluso si un servidor individual falla.

Fase 2: La Solución Cloud Native (Terraform)

Basándose en el análisis anterior, se decidió refactorizar la solución hacia una arquitectura distribuida y automatizada. Se optó por **Terraform** para garantizar la replicabilidad del entorno y se migró a un modelo sin IPs públicas fijas en los servidores, delegando el acceso externo exclusivamente al Balanceador de Carga para mejorar la seguridad y eficiencia.

Es importante señalar que, aunque el modelo *Pay-as-you-go* permite flexibilidad, la arquitectura distribuida final conlleva un coste base superior al modelo monolítico debido a la redundancia de componentes (RDS Multi-AZ, múltiples instancias EC2, EFS y ALB). Sin embargo, este incremento en el coste se justifica plenamente por la eliminación del Punto Único de Fallo y la capacidad de auto-recuperación, características indispensables para un entorno educativo crítico.

1.3. Alcance

El proyecto abarca desde la configuración inicial de la cuenta de AWS (activación de región)

hasta el despliegue de una infraestructura tolerante a fallos, incluyendo la automatización de la instalación de Moodle mediante scripts de arranque (*User Data*).

El alcance actual se centra en la infraestructura HTTP (puerto 80) para validar la arquitectura de Alta Disponibilidad. La implementación de la capa de seguridad HTTPS (SSL/TLS) mediante AWS Certificate Manager y la configuración de dominios personalizados se contemplan como pasos inmediatos para una futura puesta en producción, pero quedan fuera de la validación funcional de este TFG.

2. OBJETIVOS

2.1. Objetivo General

Diseñar, codificar e implementar una infraestructura de tres capas en AWS (Región España) capaz de alojar Moodle con tolerancia a fallos, escalado automático y persistencia de datos desacoplada, optimizando costes y rendimiento.

2.2. Objetivos Específicos

1. **Optimización Financiera (FinOps):** Eliminar el uso de IPs públicas ociosas y ajustar el dimensionamiento de las instancias (t3.micro) basándose en métricas reales.
2. **Gestión de Infraestructura Local:** Habilitar y operar en la región eu-south-2 (España) para minimizar la latencia y cumplir con normativas de datos.
3. **Automatización (IaC):** Codificar el 100% de la infraestructura con Terraform, evitando configuraciones manuales en consola ("ClickOps").
4. **Alta Disponibilidad:** Implementar un Grupo de Autoescalado (ASG) que garantice un mínimo de dos servidores activos en diferentes zonas de disponibilidad.
5. **Persistencia:** Desacoplar la base de datos (RDS) y los archivos (EFS) de la capa de cómputo.

3. ESTADO DEL ARTE Y TECNOLOGÍAS

Para la consecución del proyecto se han seleccionado tecnologías estándar en la industria *Cloud Computing*:

- **Amazon Web Services (AWS):** Proveedor de nube líder. Se han utilizado servicios gestionados (PaaS) como RDS para reducir la carga administrativa de mantenimiento de bases de datos.
- **Terraform (HashiCorp):** Herramienta de orquestación de infraestructura agnóstica. Permite definir la topología mediante lenguaje declarativo (HCL).
- **Moodle:** Plataforma LMS de código abierto, basada en PHP y MySQL.
- **Arquitectura Multi-AZ:** Diseño que distribuye los recursos en centros de datos físicamente separados (Zonas de Disponibilidad) para tolerar desastres físicos (incendios, cortes de luz) en una ubicación.

3.4. Requisitos de Dimensionamiento para Moodle

El cálculo de recursos para Moodle no es lineal y depende fuertemente de la concurrencia. Según las recomendaciones de rendimiento de Moodle (*Performance Recommendations*), una regla general aceptada es asignar aproximadamente **1 GB de RAM por cada 10 a 20 usuarios concurrentes** activos (no registrados).

En un entorno monolítico, el servidor debe gestionar tanto la carga web (Apache/PHP) como la base de datos (MySQL), lo que aumenta drásticamente el consumo de memoria. Al desacoplar la base de datos a un servicio externo (RDS), se libera memoria en los servidores web, permitiendo que instancias pequeñas como las t3.micro puedan manejar más tráfico de usuarios antes de requerir escalado.

4. METODOLOGÍA Y PLANIFICACIÓN

4.1. Enfoque Metodológico (DevOps & FinOps)

El desarrollo del proyecto se ha regido por una metodología híbrida que combina el ciclo de vida de desarrollo de software (SDLC) iterativo con prácticas **DevOps** y **FinOps**. Se ha adoptado un enfoque **"Cloud-Native First"**, priorizando el uso de servicios gestionados.

Las fases metodológicas se han estructurado en sprints técnicos:

1. **Fase de Descubrimiento y Viabilidad (FinOps):** Validación económica y técnica de la región eu-south-2 y detección de ineficiencias en el prototipo inicial.
2. **Fase de Diseño Arquitectónico (Well-Architected):** Diseño basado en los pilares de Fiabilidad (Multi-AZ) y Seguridad (Mínimo Privilegio).
3. **Fase de Implementación (IaC):** Desarrollo modular con Terraform, codificando componentes de red, datos y computación de forma aislada.
4. **Fase de Validación y Resiliencia (Chaos Engineering):** Inyección controlada de fallos para medir la capacidad de recuperación del sistema.

4.2. Fases del Proyecto

1. **Investigación y Preparación (Junio - Julio 2025):** Selección de la región y gestión de cuotas.
2. **Prototipado (Agosto - Noviembre 2025):** Despliegue manual y pruebas de rendimiento.
3. **Análisis y Refactorización (Diciembre 2025):** Detección de sobrecostes y diseño de la arquitectura final.
4. **Implementación IaC (Enero 2026):** Codificación en Terraform y pruebas de validación.

4.3. Gestión del Tenant: Activación de Región Local

Un requisito crítico fue el uso de la región **Europe (Spain) eu-south-2**. Al ser una región de reciente apertura ("Opt-in Region"), se encontraba deshabilitada por defecto en la cuenta de

AWS. Fue necesario un procedimiento administrativo de activación manual antes de poder desplegar recursos.

[PEGAR IMAGEN AQUÍ]

Figura 1: Captura del panel de AWS mostrando la región pasando de Disabled a Enabled.

5. DESARROLLO E IMPLEMENTACIÓN TÉCNICA

5.1. Análisis FinOps del Prototipo Inicial

Durante la fase de prototipado, se monitorizó el coste mediante **AWS Cost Explorer**. Se observó un gasto recurrente atribuible a las direcciones IPv4 públicas reservadas para el servidor monolítico.

Tras identificar esta ineficiencia, se procedió a liberar la IP pública a principios de Diciembre. La gráfica de costes muestra una reducción drástica del gasto, validando la necesidad de una arquitectura que no dependa de IPs fijas por instancia, sino de un Balanceador de Carga compartido.

[PEGAR IMAGEN AQUÍ]

Figura 2: Gráfica de costes mostrando la bajada en diciembre tras la liberación de la IP Pública.

5.2. Diseño de la Arquitectura de Red (VPC)

Mediante el archivo network.tf y main.tf, se definió una Virtual Private Cloud (VPC) con direccionamiento 10.0.0.0/16.

Justificación del CIDR /16:

Se ha optado por el estándar de AWS (/16, que ofrece 65.536 direcciones) para la VPC principal. Esta decisión de diseño, que no conlleva costes adicionales, garantiza la escalabilidad futura del proyecto, permitiendo la creación de nuevas subredes para backups, VPNs o entornos de pruebas sin riesgo de agotamiento de direcciones IP, siguiendo las mejores prácticas de arquitectura de red en la nube.

La red se segmentó en cuatro subredes (/24) distribuidas en dos Zonas de Disponibilidad (eu-south-2a y eu-south-2b):

- **Subredes Públicas:** Alojamiento del Balanceador de Carga (ALB).
- **Subredes Privadas:** Alojamiento de la Base de Datos RDS y Puntos de Montaje EFS.

5.3. Capa de Persistencia de Datos (storage.tf)

Para garantizar que los datos sobreviven a la destrucción de los servidores:

- **Base de Datos:** Se desplegó **Amazon RDS** con motor MySQL 8.0. Se configuró un db_subnet_group para aislarla en las subredes privadas, sin acceso directo a internet.
- **Sistema de Archivos:** Se implementó **Amazon EFS** (NFSv4). Este sistema permite que múltiples servidores web monten simultáneamente el directorio /var/www/html/moodle/moodledata, compartiendo los archivos de los cursos en tiempo real.

5.4. Capa de Computación y Automatización (asg.tf, compute.tf)

Se procedió a migrar desde un modelo de servidores estáticos ("mascotas"), que requieren mantenimiento individual, hacia un modelo de **instancias efímeras** gestionadas automáticamente. Este proceso se articuló en dos etapas clave:

1. **Creación de la Imagen Base (Golden AMI) desde el Historial:**
   A diferencia de usar una imagen genérica, se desplegó una primera instancia EC2 "piloto" mediante Terraform. Sobre esta instancia, se ejecutaron manualmente una serie de comandos de sistema (instalación de LAMP, optimización de PHP, montaje de EFS) para configurar el entorno Moodle perfecto.
   Estos comandos, extraídos del historial de la terminal (history) de la sesión de configuración, sirvieron para validar la "receta" exacta de instalación. Una vez validado el servidor piloto, se generó una **Amazon Machine Image (AMI)** personalizada a partir de él. Esta "Golden AMI" contiene el sistema operativo y la aplicación Moodle ya preconfigurados, lista para ser clonada masivamente.
2. **Launch Template:** Define la configuración de las instancias (t3.micro) y los Roles de IAM necesarios para acceder a los servicios de AWS. Se configura para usar la AMI personalizada generada en el paso anterior.
3. **User Data (Automatización Final):** Se desarrolló un script bash que se ejecuta al arranque de cada instancia clonada. Este script es el encargado de las tareas de "última milla": montar automáticamente el disco EFS compartido usando el DNS regional y ajustar permisos, garantizando que cualquier nueva instancia se conecte a los datos existentes sin intervención humana.

[PEGAR IMAGEN AQUÍ]

Figura 3: Diagrama del flujo de creación de la AMI y automatización.

5.5. Distribución de Tráfico y Seguridad

- **Balanceador (ALB):** Actúa como punto único de entrada (Puerto 80). Distribuye el tráfico entre las zonas A y B.
- **Security Groups:** Se implementó una política de "Cadena de Confianza":
  - El Servidor Web solo acepta tráfico del Balanceador.
  - La Base de Datos solo acepta tráfico del Servidor Web.
  - Nadie puede acceder directamente por IP a los recursos internos.

[PEGAR IMAGEN AQUÍ]

Figura 4: Diagrama de topología final en Mermaid.

6. RESULTADOS Y VALIDACIÓN

Se realizaron pruebas de estrés ("Chaos Testing") para validar la robustez del sistema:

6.1. Pruebas de Despliegue Automatizado

Se verificó que la ejecución del comando terraform apply es capaz de levantar la infraestructura completa (VPC, Subredes, RDS, EFS, ALB y ASG) en menos de 10 minutos, garantizando un tiempo de puesta en marcha (Time-to-Market) mínimo frente a las horas requeridas en el despliegue manual.

6.2. Fase de Validación y Resiliencia (Chaos Engineering)

Se implementó un protocolo de "Inyección controlada de fallos" para medir la capacidad de recuperación del sistema ante desastres.

Prueba de "El Exterminador" (Instance Termination):

Se terminó manualmente una instancia EC2 activa desde la consola de AWS simulando un fallo crítico de hardware.

- **Comportamiento observado:** El Application Load Balancer detectó el fallo mediante sus *Health Checks* y dejó de enviar tráfico a la instancia afectada inmediatamente. Simultáneamente, el Auto Scaling Group detectó que la capacidad saludable (1 instancia) era inferior a la capacidad deseada (2 instancias) y aprovisionó una nueva instancia automáticamente.
- **Resultado:** El servicio Moodle se mantuvo operativo gracias a la instancia redundante en la otra zona, y la capacidad total se recuperó automáticamente en menos de 3 minutos, validando la alta disponibilidad del diseño.

6.3. Validación de Persistencia

A pesar de la destrucción y recreación de instancias durante las pruebas de caos, se verificó que los cursos creados y los archivos subidos a Moodle permanecieron intactos y accesibles desde las nuevas instancias, confirmando el correcto funcionamiento del desacoplamiento de datos en RDS y EFS.

7. CONCLUSIONES Y LÍNEAS FUTURAS

El desarrollo de este TFG permite extraer las siguientes conclusiones:

1. **Automatización como Estándar:** El uso de Terraform ha transformado una tarea manual propensa a errores en un proceso de despliegue fiable y repetible en cuestión de

minutos.
2. **Importancia del Análisis de Costes:** La monitorización activa (FinOps) permitió pivotar la arquitectura hacia una solución más eficiente, eliminando el coste de IPs públicas innecesarias.
3. **Resiliencia Real:** Se ha demostrado que una arquitectura de tres capas bien diseñada en AWS es capaz de autorrepararse, garantizando la continuidad del servicio educativo incluso ante fallos graves de infraestructura.

Líneas de Trabajo Futuro:

Para llevar este proyecto a un entorno de producción final, se proponen las siguientes mejoras, que quedan fuera del alcance del presente prototipo académico:

- Implementación de HTTPS/SSL en el Balanceador de Carga mediante AWS Certificate Manager.
- Configuración de un Bastion Host para acceso SSH seguro sin exponer puertos en las instancias de aplicación.
- Implementación de un sistema de Backups automáticos de EFS y RDS con retención definida.

8. BIBLIOGRAFÍA Y REFERENCIAS

Normativa y Estándares:

- **Agencia Española de Protección de Datos (AEPD).** (2023). *Guía para clientes que contraten servicios de Cloud Computing*. Madrid: AEPD. (Fundamental para justificar la región España y RGPD).
- **Esquema Nacional de Seguridad (ENS).** (2022). *Real Decreto 311/2022*. (Marco de referencia para la seguridad de la información).

Documentación Técnica Oficial:

- **Amazon Web Services.** (2025). *Amazon Elastic File System (EFS) User Guide*. https://www.google.com/search?q=https://docs.aws.amazon.com/efs/
- **Amazon Web Services.** (2025). *Amazon EC2 Auto Scaling User Guide*. https://www.google.com/search?q=https://docs.aws.amazon.com/autoscaling/ec2/userguide/
- **HashiCorp.** (2025). *Terraform: AWS Provider Documentation*. https://www.google.com/search?q=https://registry.terraform.io/providers/hashicorp/aws/latest/docs

Recursos de Moodle y Bitnami:

- **Moodle HQ.** (2024). *Performance recommendations: Hardware configuration*. https://docs.moodle.org/en/Performance_recommendations
- **Moodle HQ.** (2022). *User site capacities*. https://www.google.com/search?q=https://docs.moodle.org/en/User_site_capacities
- **Bitnami.** (2023). *Bitnami Moodle Stack Architecture Documentation*.

9. ANEXOS

Anexo I: Estructura del Código Terraform

Descripción de la modularización del proyecto para facilitar el mantenimiento y la escalabilidad:

```
TFG-Bytemind-IaC/
├── main.tf          # Configuración de Red VPC y Gateway
├── variables.tf     # Parametrización (Región, CIDRs, Tipos de instancia)
├── security.tf      # Grupos de Seguridad (Firewall)
├── storage.tf       # Bases de Datos RDS y Sistema de Archivos EFS
├── compute.tf       # Roles de IAM y Permisos
├── asg.tf           # Auto Scaling Group y Launch Templates
├── alb.tf           # Balanceador de Carga y Target Groups
└── outputs.tf       # Salidas de consola (DNS, Endpoints)
```

Anexo II: Script de Automatización (User Data) Completo

Este script refleja la complejidad de la automatización implementada. Se inyecta en cada instancia al nacer y realiza la instalación completa del stack LAMP, la configuración de PHP y el montaje del disco compartido EFS.

Nota Técnica (Aprovisionamiento vs Configuración):

Este script de user_data se encarga del aprovisionamiento del servidor (Sistema Operativo, Paquetes, Montaje de discos). **La configuración final de la aplicación Moodle (conexión a la base de datos) NO se realiza en este script.** Dicha configuración se delega al instalador web de Moodle en el primer acceso a través del navegador. Esta decisión de diseño se ha tomado para evitar almacenar credenciales de base de datos en texto plano dentro del código de infraestructura, mejorando la seguridad del despliegue en esta fase del proyecto.

```
#!/bin/bash
# Activar modo "debug" para ver todo lo que pasa en el log del sistema
set -x
exec > >(tee /var/log/user-data.log|logger -t user-data -s 2>/dev/console) 2>&1

echo "--- INICIO DE INSTALACIÓN AUTOMÁTICA DE MOODLE ---"

# 1. Actualizar e instalar dependencias (Incluimos 'awscli' para buscar el EFS)
export DEBIAN_FRONTEND=noninteractive
apt-get update
apt-get upgrade -y
```

```
apt-get install -y apache2 nfs-common git unzip mariadb-client awscli
apt-get install -y php libapache2-mod-php php-cli php-mysql php-gd php-xml php-curl
php-mbstring php-zip php-intl php-soap php-xmlrpc php-bcmath

# 2. Descubrimiento y Montaje del EFS (El truco DevOps)
echo "Buscando el ID del EFS..."
# Usamos la CLI de AWS para encontrar el ID basado en el nombre del proyecto y la región
EFS_ID=$(aws efs describe-file-systems --region ${var.aws_region} --query
"FileSystems[?Name=='${var.project_name}-EFS'].FileSystemId" --output text)
EFS_DNS="${EFS_ID}.efs.${var.aws_region}.amazonaws.com"
echo "EFS encontrado: $EFS_DNS"

mkdir -p /var/www/moodledata

# Añadir al fstab para persistencia en reinicios
echo "$EFS_DNS:/ /var/www/moodledata nfs4
nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=2,noresvport,_netdev 0 0"
>> /etc/fstab

# Montar todo lo que hay en fstab ahora mismo
mount -a

# Permisos de datos
chown -R www-data:www-data /var/www/moodledata
chmod -R 770 /var/www/moodledata

# 3. Descargar Moodle
cd /var/www/html
rm index.html
git clone -b MOODLE_403_STABLE git://git.moodle.org/moodle.git .

# Permisos del código
chown -R www-data:www-data /var/www/html
chmod -R 755 /var/www/html

# 4. Configuración de PHP (Optimización para Moodle)
PHP_INI="/etc/php/8.1/apache2/php.ini"
echo "max_input_vars = 5000" >> $PHP_INI
echo "memory_limit = 256M" >> $PHP_INI
echo "max_execution_time = 60" >> $PHP_INI

# 5. Reiniciar Apache para aplicar todo
systemctl restart apache2
```

```
echo "--- FIN DE INSTALACIÓN AUTOMÁTICA ---"
```

Anexo III: Glosario de Términos Cloud

- **ASG (Auto Scaling Group):** Servicio de AWS que monitoriza las aplicaciones y ajusta automáticamente la capacidad para mantener un rendimiento constante y predecible al menor coste posible.
- **ALB (Application Load Balancer):** Distribuye el tráfico entrante de aplicaciones entre múltiples destinos, como instancias EC2, en varias zonas de disponibilidad. En esta arquitectura, el ALB actúa como el único punto de entrada público, ocultando la topología interna.
- **EFS (Elastic File System):** Sistema de archivos NFS elástico y sin servidor que se puede montar en múltiples instancias EC2 simultáneamente.
- **RDS (Relational Database Service):** Servicio de base de datos gestionado que facilita la configuración, operación y escalado de una base de datos relacional en la nube.
- **VPC (Virtual Private Cloud):** Red virtual dedicada en la cuenta de AWS, aislada lógicamente de otras redes virtuales.
- **IaC (Infrastructure as Code):** Práctica de gestionar y aprovisionar infraestructura a través de código en lugar de procesos manuales.
- **FinOps:** Práctica cultural y operativa de gestión financiera en la nube, promoviendo la responsabilidad compartida sobre los costes.

Anexo IV: Comandos Base del Prototipo Manual (Fase 1)

Registro de los comandos ejecutados manualmente en la terminal durante la Fase 1 para configurar el prototipo Bitnami y validar la conectividad a la base de datos local. Esta fase sirvió para identificar la necesidad de desacoplar los servicios.

```
# Conexión SSH al servidor prototipo
ssh -i "bitnami-key.pem" admin@ip-publica-fija

# Verificación de servicios monolíticos
sudo /opt/bitnami/ctlscript.sh status

# Conexión manual a la base de datos local (Mariadb)
mysql -u root -p
# (Password hardcodeada en el servidor)

# Configuración de IP estática (Causa del sobrecoste detectado)
# Se editó manualmente el archivo de configuración de red para asociar la Elastic IP
```

Anexo V: Evidencia del Despliegue Monolítico (Fase 1)

Descripción:

Este anexo contiene evidencia documental de la primera fase del proyecto, donde se desplegó una solución basada en la imagen de mercado Bitnami Moodle.

Contenido:

1. **Selección de AMI:**
   [PEGAR IMAGEN AQUÍ]
   Figura 5: Selección de la imagen Bitnami Certified by Automattic.
2. **Configuración Manual:**
   [PEGAR IMAGEN AQUÍ]
   Figura 6: Conexión SSH y gestión de credenciales locales (cat bitnami_credentials).
3. **Base de Datos Local:**
   [PEGAR IMAGEN AQUÍ]
   Figura 7: Conexión a la base de datos MariaDB local (localhost), evidenciando el SPOF.
4. **Gestión de Red:**
   [PEGAR IMAGEN AQUÍ]
   Figura 8: Asignación de IP Elástica (EIP), identificada como ineficiencia de costes.