■ Proyecto Flask - Código Python (Parte 2)

Total de archivos: 8

Archivo 1: ./routes/monitor_routes.py

```
from flask import jsonify■
import asyncio■
import threading■
from datetime import datetime■
from services.maas_client import obtener_maquinas_nuevas■
from services.telegram_service import enviar_notificacion_telegram■
■
def setup_monitor_routes(app, monitor):■
    """Configura las rutas del monitor"""■
■
    @app.route("/monitor/start", methods=["POST"])■
    def iniciar_monitor():■
        """Endpoint para iniciar el monitoreo"""■
        try:■
            if not monitor.monitoreo_activo:■
                threading.Thread(target=lambda: asyncio.run(monitor.iniciar_monitoreo()), daemon=True).start()■
                return jsonify({■
                    "estado": "Monitoreo iniciado",■
                    "mensaje": "El Monitoreo de máquinas MAAS iniciado correctamente"■
                })■
            else:■
                return jsonify({■
                    "estado": "El monitoreo ya está activo",■
                    "mensaje": "El monitoreo ya se encuentra en ejecución"■
                })■
        except Exception as e:■
            return jsonify({'error': str(e)}), 500■
■
    @app.route("/monitor/stop", methods=["POST"])■
    def detener_monitor():■
        """Endpoint para detener el monitoreo"""■
        try:■
            monitor.detener_monitoreo()■
            return jsonify({■
                "estado": "Monitoreo detenido",■
                "mensaje": "El Monitoreo de máquinas detenido correctamente"■
            })■
        except Exception as e:■
            return jsonify({'error': str(e)}), 500■
■
    @app.route("/monitor/status", methods=["GET"])■
    def estado_monitor():■
        """Endpoint para ver el estado del monitoreo"""■
        return jsonify({■
            "monitoreo_activo": monitor.monitoreo_activo,■
            "maquinas_monitoreadas": len(monitor.estados_anteriores),■
            "intervalo_segundos": monitor.intervalo,■
            "estado": "Activo" if monitor.monitoreo_activo else "Inactivo"■
        })■
■
    @app.route("/monitor/check-new", methods=["POST"])■
    @app.route("/debug/power/<system_id>", methods=["GET"])■
    def debug_power_route(system_id):■
        """Endpoint para debug de power parameters"""■
        try:■
            from services.maas_client import debug_power_parameters■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            resultado = loop.run_until_complete(debug_power_parameters(system_id))■
            return jsonify({"debug_power": resultado})■
        except Exception as e:■
            return jsonify({'error': str(e)}), 500■
    def verificar_maquinas_nuevas():■
        """Endpoint para verificar máquinas nuevas manualmente"""■
        try:■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            maquinas_nuevas = loop.run_until_complete(obtener_maquinas_nuevas())■
            ■
            # Notificar por Telegram si hay máquinas nuevas■
            if maquinas_nuevas:■
                for maquina in maquinas_nuevas:■
                    mensaje = (■
                        f"■ <b>NUEVA MÁQUINA DETECTADA (Manual)</b>\n\n"■
                        f"■ <b>Nombre:</b> {maquina['hostname']}\n"■
```

```
                f"■ <b>IP:</b> {maquina['ip']}\n"■
                f"■ <b>ID:</b> {maquina['system_id']}\n"■
                f"■ <b>Estado:</b> {maquina['status']}\n\n"■
                f"■ <i>Esta máquina necesita commissioning y deploy</i>"■
            )■
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje)).start()■
    ■
    return jsonify({■
        "maquinas_nuevas": maquinas_nuevas,■
        "total": len(maquinas_nuevas),■
        "mensaje": f"Se encontraron {len(maquinas_nuevas)} máquinas nuevas",■
        "timestamp": datetime.now().isoformat()■
    })■
except Exception as e:■
    return jsonify({'error': str(e)}), 500
```

Archivo 2: ./routes/chat_routes.py

```
from flask import jsonify, request■
import asyncio■
from services.chat_service import responder_pregunta■
■
def setup_chat_routes(app):■
    @app.route("/preguntar", methods=["POST"])■
    def preguntar():■
        try:■
            data = request.get_json()■
            if not data:■
                return jsonify({'respuesta': 'No se recibieron datos JSON'}), 400■
■
            pregunta = data.get("pregunta", '')■
            if not pregunta.strip():■
                return jsonify({'respuesta': 'Por favor, ingresa una pregunta.'}), 400■
■
            print(f"Pregunta recibida: {pregunta}")■
■
            loop = asyncio.new_event_loop()■
            asyncio.set_event_loop(loop)■
            respuesta = loop.run_until_complete(responder_pregunta(pregunta))■
            print(f"Respuesta generada: {respuesta}")■
            return jsonify({'respuesta': respuesta})■
■
        except Exception as e:■
            print(f"Error en endpoint /preguntar: {e}")■
            import traceback■
            traceback.print_exc()■
            return jsonify({'respuesta': f'Error procesando la pregunta: {str(e)}'}), 500
```

Archivo 3: ./models/monitor.py

```
import asyncio■
from datetime import datetime■
from services.maas_client import obtener_estado_actual, obtener_maquinas_nuevas■
from services.telegram_service import enviar_notificacion_telegram■
■
class MonitorMaquinas:■
    def __init__(self):■
        self.estados_anteriores = {}■
        self.maquinas_nuevas_detectadas = set()  # Para trackear máquinas nuevas ya notificadas■
        self.monitoreo_activo = False■
        self.intervalo = 30■
    ■
    def detectar_cambios(self, estados_actuales):■
        cambios = []■
        ■
        for hostname, estado_actual in estados_actuales.items():■
            estado_anterior = self.estados_anteriores.get(hostname, {})■
            ■
            if hostname not in self.estados_anteriores:■
                cambios.append(f"■ <b>Nueva máquina detectada:</b> {hostname} ({estado_actual['ip']}) - Estado: {estad
            else:■
                if estado_anterior.get('power_state') != estado_actual['power_state']:■
                    if estado_actual['power_state'] == 'on':■
                        cambios.append(f"■ <b>Máquina encendida:</b> {hostname} ({estado_actual['ip']})")■
                    elif estado_actual['power_state'] == 'off':■
                        cambios.append(f"■ <b>Máquina apagada:</b> {hostname} ({estado_actual['ip']})")■
                    else:■
                        cambios.append(f"■ <b>Estado cambiado:</b> {hostname} ({estado_actual['ip']}) - Nuevo estado:
        ■
        for hostname in self.estados_anteriores:■
            if hostname not in estados_actuales:■
                cambios.append(f"■ <b>Máquina desaparecida:</b> {hostname}")■
        ■
        return cambios■
    ■
    async def verificar_maquinas_nuevas(self):■
        """Verifica y notifica sobre máquinas nuevas, abortando commissioning automático"""■
        try:■
            maquinas_nuevas = await obtener_maquinas_nuevas()■
            ■
            for maquina in maquinas_nuevas:■
                maquina_id = maquina['system_id']■
                ■
                # Si no hemos procesado esta máquina nueva aún■
                if maquina_id not in self.maquinas_nuevas_detectadas:■
                    ■
                    print(f"■ Máquina nueva detectada: {maquina['hostname']} (ID: {maquina_id})")■
                    ■
                    # === ABORTAR COMMISSIONING AUTOMÁTICO ===■
                    from services.maas_client import abortar_commissioning■
                    resultado_abort = await abortar_commissioning(maquina_id)■
                    ■
                    # Construir mensaje de notificación■
                    mensaje = (■
                        f"■ <b>NUEVA MÁQUINA DETECTADA</b>\n\n"■
                        f"■ <b>Nombre MAAS:</b> {maquina['hostname']}\n"■
                        f"■ <b>IP:</b> {maquina['ip']}\n"■
                        f"■ <b>ID:</b> {maquina['system_id']}\n"■
                        f"■ <b>Estado:</b> {maquina['status']}\n\n"■
                        f"■ <b>Acción realizada:</b> Commissioning automático abortado\n"■
                        f"■ <b>Resultado:</b> {resultado_abort}\n\n"■
                        f"■ <b>Para configurar el power Virsh:</b>\n"■
                        f"Escribe en el chat: <code>configurar máquina</code>\n\n"■
                        f"■ {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"■
                    )■
                    ■
                    # Enviar notificación por Telegram■
                    if enviar_notificacion_telegram(mensaje):■
                        print(f"■ Notificación enviada: {maquina['hostname']}")■
                        self.maquinas_nuevas_detectadas.add(maquina_id)■
                    else:■
                        print(f"■ Error enviando notificación: {maquina['hostname']}")■
            ■
        except Exception as e:■
            print(f"■ Error en verificación de máquinas nuevas: {e}")■
```

```
■
async def verificar_estados(self):■
    try:■
        estados_actuales = await obtener_estado_actual()■
        ■
        # Verificar máquinas nuevas primero■
        await self.verificar_maquinas_nuevas()■
        ■
        if self.estados_anteriores:■
            cambios = self.detectar_cambios(estados_actuales)■
            ■
            for cambio in cambios:■
                mensaje_completo = f"<b>■ Notificación MAAS</b>\n{cambio}\n■ {datetime.now().strftime('%Y-%m-%d %
                if enviar_notificacion_telegram(mensaje_completo):■
                    print(f"■ Notificación enviada: {cambio}")■
                else:■
                    print(f"■ Error enviando notificación: {cambio}")■
                await asyncio.sleep(1)■
        ■
        self.estados_anteriores = estados_actuales■
        ■
    except Exception as e:■
        print(f"■ Error en verificación de estados: {e}")■
■
async def iniciar_monitoreo(self):■
    self.monitoreo_activo = True■
    print("■ Iniciando monitoreo de máquinas MAAS...")■
    ■
    try:■
        # Obtener estado inicial y máquinas nuevas■
        self.estados_anteriores = await obtener_estado_actual()■
        await self.verificar_maquinas_nuevas()  # Verificar máquinas nuevas al iniciar■
        print(f"■ Estado inicial capturado: {len(self.estados_anteriores)} máquinas")■
    except Exception as e:■
        print(f"■ Error en verificación inicial: {e}")■
    ■
    while self.monitoreo_activo:■
        try:■
            await self.verificar_estados()■
            await asyncio.sleep(self.intervalo)■
        except Exception as e:■
            print(f"■ Error en bucle de monitoreo: {e}")■
            await asyncio.sleep(self.intervalo)■
■
def detener_monitoreo(self):■
    self.monitoreo_activo = False■
    print("■ Monitoreo detenido")
```

Archivo 4: ./utils/helpers.py

```
def serializar_objeto_simple(obj):■
    if obj is None:■
        return None■
    elif isinstance(obj, (str, int, float, bool)):■
        return obj■
    elif isinstance(obj, list):■
        return [serializar_objeto_simple(item) for item in obj]■
    elif isinstance(obj, dict):■
        return {key: serializar_objeto_simple(value) for key, value in obj.items()}■
    elif hasattr(obj, '__dict__'):■
        result = {}■
        for key, value in obj.__dict__.items():■
            if not key.startswith('_'):■
                try:■
                    result[key] = serializar_objeto_simple(value)■
                except:■
                    result[key] = str(value)■
        return result■
    else:■
        return str(obj)
```

Archivo 5: ./services/chat_service.py

```
import asyncio■
import re■
from config import CORTESIAS■
from services.maas_client import listar_maquinas, listar_subredes, encender_maquina, apagar_maquina, buscar_maquina_po
from services.gemini_service import generar_respuesta_gemini■
from services.maas_client import obtener_maquinas_nuevas, configurar_power_virsh■
from services.maas_client import listar_maquinas_para_commissioning, ejecutar_commissioning■
from services.maas_client import listar_maquinas_para_deploy, ejecutar_deploy■
■
# Estados de conversación para configuración interactiva:■
estados_conversacion = {}■
■
class EstadoConfiguracion:■
    def __init__(self, system_id, hostname_maas, ip_maquina):■
        self.system_id = system_id■
        self.hostname_maas = hostname_maas■
        self.ip_maquina = ip_maquina■
        self.paso_actual = "esperando_nombre"■
        self.vm_id_asignado = None■
■
async def responder_pregunta(pregunta):■
    try:■
        pregunta_lower = pregunta.lower()■
■
        # Filtrar cortesias■
        if any(c in pregunta_lower for c in CORTESIAS):■
            return "[De nada] ■"■
■
        # ====== COMANDOS DE COMMISSIONING - DEBEN IR PRIMERO ========■
■
        # Comando para ejecutar commissioning CON NÚMERO (más específico primero)■
        if any(palabra in pregunta_lower for palabra in ['commissioning', 'comisionar']) and any(c.isdigit() for c in
            try:■
                # Extraer número de la pregunta■
                numeros = [int(s) for s in pregunta.split() if s.isdigit()]■
                if not numeros:■
                    return "■ Por favor, especifica el número de la máquina. Ejemplo: 'commissioning 1'"■
■
                numero_maquina = numeros[0]■
                maquinas = await listar_maquinas_para_commissioning()■
■
                if numero_maquina < 1 or numero_maquina > len(maquinas):■
                    return f"■ Número inválido. Por favor elige un número entre 1 y {len(maquinas)}"■
■
                maquina_seleccionada = maquinas[numero_maquina - 1]■
                resultado = await ejecutar_commissioning(maquina_seleccionada['system_id'])■
■
                if resultado['success']:■
                    respuesta = f"■ **COMMISSIONING INICIADO**\n\n"■
                    respuesta += f"**Máquina:** {maquina_seleccionada['hostname']}\n"■
                    respuesta += f"**IP:** {maquina_seleccionada['ip']}\n"■
                    respuesta += f"**ID:** {maquina_seleccionada['system_id']}\n\n"■
                    respuesta += "■ El proceso de commissioning ha comenzado. Esto puede tomar varios minutos.\n"■
                    respuesta += "■ Puedes monitorear el progreso en el dashboard de MAAS."■
■
                else:■
                    respuesta = f"■ **ERROR EN COMMISSIONING**\n\n{resultado['message']}"■
■
                return respuesta■
■
            except Exception as e:■
                return f"■ Error al ejecutar commissioning: {str(e)}"■
■
        # Comando para listar máquinas para commissioning (sin número)■
        elif any(palabra in pregunta_lower for palabra in ['commissioning', 'comisionar', 'maquinas para commissioning
            try:■
                maquinas = await listar_maquinas_para_commissioning()■
■
                if not maquinas:■
                    return "■ No hay máquinas disponibles para commissioning. Todas las máquinas están en estado 'Depl
■
                respuesta = "■ **MÁQUINAS DISPONIBLES PARA COMMISSIONING**\n\n"■
                for i, maquina in enumerate(maquinas, 1):■
                    respuesta += f"{i}. **{maquina['hostname']}**\n"■
                    respuesta += f"   IP: {maquina['ip']}\n"■
```

```
                respuesta += f"   Estado: {maquina['status']}\n"
                respuesta += f"   ID: {maquina['system_id']}\n\n"

            respuesta += "\n **Para ejecutar commissioning:**\n"
            respuesta += "Escribe: 'commissioning [número]' o 'comisionar [número]'\n"
            respuesta += "Ejemplo: 'commissioning 1' para ejecutar en la primera máquina"

            return respuesta

        except Exception as e:
            return f"\n Error al listar máquinas para commissioning: {str(e)}"

    # ====== COMANDOS DE DEPLOY ========

    # Comando para ejecutar deploy CON NÚMERO (y posiblemente opciones)
    elif any(palabra in pregunta_lower for palabra in ['deploy', 'desplegar']) and any(c.isdigit() for c in pregun
        try:
            # Extraer número y opciones
            partes = pregunta.split()
            numeros = [int(s) for s in partes if s.isdigit()]
            if not numeros:
                return "\n Por favor, especifica el número de la máquina. Ejemplo: 'deploy 1'"

            numero_maquina = numeros[0]
            maquinas = await listar_maquinas_para_deploy()

            if numero_maquina < 1 or numero_maquina > len(maquinas):
                return f"\n Número inválido. Por favor elige un número entre 1 y {len(maquinas)}"

            # Procesar opciones si las hay
            opciones = {}
            for parte in partes:
                if ':' in parte:
                    key, value = parte.split(':', 1)
                    opciones[key.strip().lower()] = value.strip()

            maquina_seleccionada = maquinas[numero_maquina - 1]
            resultado = await ejecutar_deploy(maquina_seleccionada['system_id'], opciones)

            if resultado['success']:
                respuesta = f"\n **DEPLOY INICIADO**\n\n"
                respuesta += f"**Máquina:** {maquina_seleccionada['hostname']}\n"
                respuesta += f"**IP:** {maquina_seleccionada['ip']}\n"
                respuesta += f"**ID:** {maquina_seleccionada['system_id']}\n"

                if opciones:
                    respuesta += f"**Opciones:** {opciones}\n\n"
                else:
                    respuesta += "\n"

                respuesta += "\n El proceso de deploy ha comenzado. Esto puede tomar varios minutos.\n"
                respuesta += "\n Puedes monitorear el progreso en el dashboard de MAAS."

            else:
                respuesta = f"\n **ERROR EN DEPLOY**\n\n{resultado['message']}"

            return respuesta

        except Exception as e:
            return f"\n Error al ejecutar deploy: {str(e)}"

    # Comando para listar máquinas para deploy (sin número)
    elif any(palabra in pregunta_lower for palabra in ['deploy', 'desplegar', 'maquinas para deploy', 'máquinas pa
        try:
            maquinas = await listar_maquinas_para_deploy()

            if not maquinas:
                return "\n No hay máquinas disponibles para deploy. Las máquinas deben estar en estado 'Ready'."

            respuesta = "\n **MÁQUINAS DISPONIBLES PARA DEPLOY**\n\n"
            for i, maquina in enumerate(maquinas, 1):
                respuesta += f"{i}. **{maquina['hostname']}**\n"
                respuesta += f"   IP: {maquina['ip']}\n"
                respuesta += f"   Estado: {maquina['status']}\n"
                respuesta += f"   SO: {maquina['osystem']} | Arquitectura: {maquina['architecture']}\n"
                respuesta += f"   RAM: {maquina['memory_gb']} GB | CPUs: {maquina['cpu_count']}\n"
                respuesta += f"   ID: {maquina['system_id']}\n\n"
```

```
            respuesta += "■ **Para ejecutar deploy:**\n"■
            respuesta += "Escribe: 'deploy [número]' o 'desplegar [número]'\n"■
            respuesta += "Ejemplo: 'deploy 1' para desplegar la primera máquina\n\n"■
            respuesta += "■ **Opciones de deploy personalizado:**\n"■
            respuesta += "Puedes añadir opciones después del número:\n"■
            respuesta += "- distro:[nombre] (ej: distro:jammy)\n"■
            respuesta += "- kernel:[nombre] (ej: kernel:hwe-22.04)\n"■
            respuesta += "Ejemplo: 'deploy 1 distro:focal kernel:hwe-20.04'"■
■
            return respuesta■
■
        except Exception as e:■
            return f"■ Error al listar máquinas para deploy: {str(e)}"■
■
    # Comando para deploy con opciones específicas■
    elif any(palabra in pregunta_lower for palabra in ['deploy personalizado', 'deploy con opciones']):■
        try:■
            maquinas = await listar_maquinas_para_deploy()■
            ■
            if not maquinas:■
                return "■ No hay máquinas disponibles para deploy."■
■
            respuesta = "■ **DEPLOY PERSONALIZADO**\n\n"■
            respuesta += "**Máquinas disponibles:**\n"■
            for i, maquina in enumerate(maquinas, 1):■
                respuesta += f"{i}. {maquina['hostname']} ({maquina['ip']})\n"■
■
            respuesta += "\n■ **Para deploy personalizado:**\n"■
            respuesta += "Escribe: 'deploy [número] [opciones]'\n"■
            respuesta += "Opciones disponibles:\n"■
            respuesta += "- distro:[nombre] (ej: distro:jammy)\n"■
            respuesta += "- kernel:[nombre] (ej: kernel:hwe-22.04)\n"■
            respuesta += "Ejemplo: 'deploy 1 distro:focal kernel:hwe-20.04'"■
■
            return respuesta■
■
        except Exception as e:■
            return f"■ Error: {str(e)}"■
■
    # =========== FIN COMANDOS DE DEPLOY ==========■
■
    # DETECTAR COMANDOS DE CONTROL DE MÁQUINAS:■
    # Comando para encender máquinas:■
    elif any(palabra in pregunta_lower for palabra in ['enciendo', 'encienda', 'prende', 'prenda', 'power on', 'en
        # Extraer el identificador de la máquina:■
        identificador = None■
        # Buscar por nombre de máquina:■
        maquinas_texto = await listar_maquinas()■
■
        for linea in maquinas_texto.split('\n'):■
            if 'MÁQUINA:' in linea:■
                partes = linea.split('(')■
                if len(partes) > 1:■
                    # Extraer nombre limpio sin emojis:■
                    nombre_completo = partes[0].replace('MÁQUINA:', '').strip()■
                    nombre_maquina = re.sub(r'[^a-zA-Z0-9]', '', nombre_completo).strip()■
                    system_id = partes[1].replace(')', '').strip()■
■
                    # Coincidencia más flexible:■
                    if nombre_maquina.lower() in pregunta_lower:■
                        identificador = nombre_maquina■
                        break■
                    elif system_id.lower() in pregunta_lower:■
                        identificador = system_id■
                        break■
■
        # Buscar por ip:■
        if not identificador:■
            ip_pattern = r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b'■
            ips = re.findall(ip_pattern, pregunta)■
            if ips:■
                maquina_por_ip = await buscar_maquina_por_ip(ips[0])■
                if maquina_por_ip:■
                    identificador = maquina_por_ip.hostname■
■
        if not identificador:■
```

```
            # Si no se encontró identificador, mostrar máquinas disponibles:■
            maquinas_disponibles = []■
            for linea in maquinas_texto.split('\n'):■
                if 'MÁQUINA:' in linea:■
                    nombre_completo = linea.split('MÁQUINA:')[1].split('(')[0].strip()■
                    nombre_limpio = re.sub(r'[^a-zA-Z0-9]', '', nombre_completo).strip()■
                    maquinas_disponibles.append(nombre_limpio)■
■
            if maquinas_disponibles:■
                return f"■ ¿Qué máquina quieres encender?\n\n Máquinas disponibles:\n" + "\n".join([f"• {maquina}'
            else:■
                return "■ No se encontraron máquinas disponibles."■
■
        # Ejecutar comando de encender:■
        resultado = await encender_maquina(identificador)■
        return resultado■
■
    # Comando para apagar maquina:■
    elif any(palabra in pregunta_lower for palabra in ['apaga', 'apague', 'apagar', 'power off', 'apagado']):■
        # Extraer el identificador de la máquina:■
        identificador = None■
        # Buscar por nombre de máquina:■
        maquinas_texto = await listar_maquinas()■
■
        for linea in maquinas_texto.split('\n'):■
            if 'MÁQUINA:' in linea:■
                partes = linea.split('(')■
                if len(partes) > 1:■
                    # Extraer nombre limpio sin emojis:■
                    nombre_completo = partes[0].replace('MÁQUINA:', '').strip()■
                    nombre_maquina = re.sub(r'[^a-zA-Z0-9]', '', nombre_completo).strip()■
                    system_id = partes[1].replace(')', '').strip()■
■
                    # Coincidencia más flexible:■
                    if nombre_maquina.lower() in pregunta_lower:■
                        identificador = nombre_maquina■
                        break■
                    elif system_id.lower() in pregunta_lower:■
                        identificador = system_id■
                        break■
■
        # Buscar por ip:■
        if not identificador:■
            ip_pattern = r'\b(?:[0-9]{1,3}\.){3}[0-9]{1,3}\b'■
            ips = re.findall(ip_pattern, pregunta)■
            if ips:■
                maquina_por_ip = await buscar_maquina_por_ip(ips[0])■
                if maquina_por_ip:■
                    identificador = maquina_por_ip.hostname■
■
        if not identificador:■
            # Si no se encontró identificador, mostrar máquinas disponibles:■
            maquinas_disponibles = []■
            for linea in maquinas_texto.split('\n'):■
                if 'MÁQUINA:' in linea:■
                    nombre_completo = linea.split('MÁQUINA:')[1].split('(')[0].strip()■
                    nombre_limpio = re.sub(r'[^a-zA-Z0-9]', '', nombre_completo).strip()■
                    maquinas_disponibles.append(nombre_limpio)■
■
            if maquinas_disponibles:■
                return f"■ ¿Qué máquina quieres apagar?\n\n Máquinas disponibles:\n" + "\n".join([f"• {maquina}" f
            else:■
                return "■ No se encontraron máquinas disponibles."■
■
        # Ejecutar comando de apagar:■
        resultado = await apagar_maquina(identificador)■
        return resultado■
■
    # Comando para verificar maquinas nuevas:■
    elif any(palabra in pregunta_lower for palabra in ['maquinas nuevas', 'máquinas nuevas', 'nuevas detectadas']):
        try:■
            maquinas_nuevas = await obtener_maquinas_nuevas()■
            if not maquinas_nuevas:■
                return "■ No hay máquinas nuevas detectadas. Todas las máquinas están configuradas."■
■
            respuesta = "■ **MÁQUINAS NUEVAS DETECTADAS**\n\n"■
            for maquina in maquinas_nuevas:■
```

```
                    respuesta += (■
                        f"**Nombre:** {maquina['hostname']}\n"■
                        f"**IP:** {maquina['ip']}\n"■
                        f"**Estado:** {maquina['status']}\n"■
                        f"**ID:** {maquina['system_id']}\n\n"■
                    )■
■
                respuesta += "■■ *Estas máquinas necesitan proceso de commissioning y deploy*"■
                return respuesta■
            except Exception as e:■
                return f"■ Error al verificar máquinas nuevas: {str(e)}"■
■
        # Comando para iniciar configuración interactiva:■
        elif any(palabra in pregunta_lower for palabra in ['configurar maquina', 'configurar máquina', 'asignar nombre
            # Primero, obtener máquinas nuevas■
            maquinas_nuevas = await obtener_maquinas_nuevas()■
■
            if not maquinas_nuevas:■
                return "■ No hay máquinas nuevas para configurar."■
■
            if len(maquinas_nuevas) == 1:■
                # Si hay solo una máquina nueva, empezar configuración directa:■
                maquina = maquinas_nuevas[0]■
                estados_conversacion['usuario_actual'] = EstadoConfiguracion(■
                    maquina['system_id'],■
                    maquina['hostname'],■
                    maquina['ip']■
                )■
■
                return (■
                    f"■■ **CONFIGURAR MÁQUINA NUEVA**\n\n"■
                    f"**Información de la máquina:**\n"■
                    f"**Hostname MAAS:** {maquina['hostname']}\n"■
                    f"**IP:** {maquina['ip']}\n"■
                    f"**ID:** {maquina['system_id']}\n\n"■
                    f"**¿Cómo se llama esta máquina en Virsh?**\n"■
                    f"**Responde con el VM ID exacto (ej: maq2, servidor-web, etc.)**"■
                )■
■
            else:■
                # Si hay múltiples máquinas, listarlas■
                respuesta = "■ **MÚLTIPLES MÁQUINAS NUEVAS DETECTADAS**\n\n"■
                for i, maquina in enumerate(maquinas_nuevas, 1):■
                    respuesta += (■
                        f"{i}. **{maquina['hostname']}** (IP: {maquina['ip']})\n"■
                    )■
■
                respuesta += (■
                    f"\n**Responde con el número de la máquina que quieres configurar**\n"■
                    f"**Ejemplo: '1' para {maquinas_nuevas[0]['hostname']}**"■
                )■
■
                # Guardar estado temporal■
                estados_conversacion['maquinas_lista'] = maquinas_nuevas■
                estados_conversacion['paso'] = 'seleccionar_maquina'■
■
                return respuesta■
■
        # Manejar respuestas de configuración (cuando hay estado activo)■
        elif 'usuario_actual' in estados_conversacion:■
            estado = estados_conversacion['usuario_actual']■
            if estado.paso_actual == "esperando_nombre":■
                # El usuario está respondiendo con el nombre de la máquina■
                vm_id = pregunta.strip()■
■
                # Configurar el power Virsh con el nombre proporcionado■
                resultado_power = await configurar_power_virsh(estado.system_id, vm_id)■
■
                # Limpiar estado■
                del estados_conversacion['usuario_actual']■
                return (■
                    f"■ **CONFIGURACIÓN COMPLETADA**\n\n"■
                    f"**Máquina:** {estado.hostname_maas}\n"■
                    f"**IP:** {estado.ip_maquina}\n"■
                    f"**VM ID asignado:** {vm_id}\n\n"■
                    f"{resultado_power}\n\n"■
                    f"■ *Ahora puedes realizar el commissioning manual cuando lo necesites*"■
```

```
                )■
■
        # Manejar selección de maquina de la lista■
        elif 'maquinas_lista' in estados_conversacion and estados_conversacion.get('paso') == 'seleccionar_maquina':■
            try:■
                numero = int(pregunta.strip())■
                maquinas_lista = estados_conversacion['maquinas_lista']■
■
                if 1 <= numero <= len(maquinas_lista):■
                    maquina = maquinas_lista[numero - 1]■
■
                    # Iniciar configuración para esta máquina■
                    estados_conversacion['usuario_actual'] = EstadoConfiguracion(■
                        maquina['system_id'],■
                        maquina['hostname'],■
                        maquina['ip']■
                    )■
■
                    # Limpiar estado temporal■
                    del estados_conversacion['maquinas_lista']■
                    del estados_conversacion['paso']■
■
                    return (■
                        f"■■ **CONFIGURAR MÁQUINA NUEVA**\n\n"■
                        f"**Información de la máquina:**\n"■
                        f"**Hostname MAAS:** {maquina['hostname']}\n"■
                        f"**IP:** {maquina['ip']}\n"■
                        f"**ID:** {maquina['system_id']}\n\n"■
                        f"**¿Cómo se llama esta máquina en Virsh?**\n"■
                        f"**Responde con el VM ID exacto (ej: maq2, servidor-web, etc.)**"■
                    )■
■
                else:■
                    return f"■ Número inválido. Por favor, elige un número entre 1 y {len(maquinas_lista)}."■
■
            except ValueError:■
                return "■ Por favor, responde con un número válido."■
■
        # Consulta de subredes:■
        elif 'subred' in pregunta_lower or "subredes" in pregunta_lower:■
            subredes_texto = await listar_subredes()■
            prompt = f"""■
INFORMACIÓN DE SUBREDES EN MAAS:■
{subredes_texto}■
■
PREGUNTA DEL USUARIO: {pregunta}■
■
INSTRUCCIONES:■
- Responde en español de forma clara y organizada.■
- Usa emojis para hacerlo visual.■
- Agrupa la información de forma lógica.■
- Destaca los datos más importantes.■
- Formato: título, luego lista de subredes con sus características.■
- Responde:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
        # Consulta de máquinas (consultas informativas):■
        maquinas_texto = await listar_maquinas()■
■
        # DETECTAR TIPO DE PREGUNTA PARA ADAPTAR LA RESPUESTA:■
        if any(palabra in pregunta_lower for palabra in ['cuántas', 'cuantas', 'número', 'numero', 'total', 'cuantos']
            prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Analiza cuántas máquinas hay en total.■
- Cuenta cuántas están encendidas y cuántas apagadas.■
- Proporciona estadísticas claras.■
- Usa formato visual con emojis.■
- Incluye detalles interesantes sobre el estado general.■
■
EJEMPLO DE FORMATO:■
**Resumen del Sistema**■
```

```
**Total de máquinas**: X■
**Encendidas**: Y■
**Apagadas**: Z■
**Porcentaje activas**: W%■
■
**Lista de máquinas**: [Breve lista con nombres y estados]■
■
Responde en español:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
        elif any(palabra in pregunta_lower for palabra in ['ram', "memoria"]):■
            prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre la memoria RAM.■
- Organiza la información de forma clara.■
- Si hay máquinas específicas mencionadas, enfócate en ellas.■
- Usa emojis y formato visual.■
- Incluye totales si es relevante.■
■
Formato sugerido:■
**Información de Memoria RAM**■
■
**Total de RAM en el sistema**: X GB■
■
**Por máquina**:■
[Máquina1]: [RAM] GB■
[Máquina2]: [RAM] GB■
■
Responde en español:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
        elif any(palabra in pregunta_lower for palabra in ['almacenamiento', 'disco', 'disco duro', 'storage', 'gb']):
            prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre el almacenamiento.■
- Organiza la información de forma clara.■
- Usa emojis y formato visual.■
- Incluye totales si es relevante.■
■
Formato sugerido:■
**Información de Almacenamiento**■
**Total de almacenamiento**: X GB■
■
**Por máquina**:■
[Máquina1]: [Almacenamiento] GB■
[Máquina2]: [Almacenamiento] GB■
■
Responde en español:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
        elif any(palabra in pregunta_lower for palabra in ['cpu', 'procesador', 'núcleo', 'núcleos', 'procesadores']):
            prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre los CPUs/procesadores.■
- Organiza la información de forma clara.■
- Usa emojis y formato visual.■
- Incluye totales si es relevante.■
■
Formato sugerido:■
```

```
**Información de Procesadores**■
**Total de núcleos en el sistema**: X■
■
**Por máquina**:■
[Máquina1]: [CPUs] núcleos■
[Máquina2]: [CPUs] núcleos■
■
Responde en español:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
        elif any(palabra in pregunta_lower for palabra in ['encend', 'apag', 'power', 'on', 'off', 'estado']):■
            prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre el estado de encendido/apagado.■
- Organiza por estado (encendidas primero, luego apagadas).■
- Usa emojis visuales (■ para encendidas, ■ para apagadas).■
- Sé claro y conciso.■
■
Formato sugerido:■
**Estado de las Máquinas**■
**Encendidas** (X):■
[Máquina1] ([IP])■
[Máquina2] ([IP])■
■
**Apagadas** (Y):■
[Máquina3] ([IP])■
[Máquina4] ([IP])■
■
Responde en español:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
        elif any(palabra in pregunta_lower for palabra in ['ip', 'dirección', 'direccion', 'red', 'network']):■
            prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Responde ÚNICAMENTE sobre las direcciones IP.■
- Organiza la información de forma clara.■
- Incluye el estado de cada máquina.■
■
Formato sugerido:■
**Direcciones IP del Sistema**■
■
**Máquinas y sus IPs**■
[Máquina1]: [IP]■
[Máquina2]: [IP]■
[Máquina3]: [IP]■
■
Responde en español:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
        elif any(palabra in pregunta_lower for palabra in ['información', 'info', 'detalles', 'resumen', 'todo', 'gene
            prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Proporciona un resumen COMPLETO pero BIEN ESTRUCTURADO de todas las máquinas.■
- Agrupa por estado (encendidas primero).■
- Para CADA máquina, incluye: estado, IP, RAM, almacenamiento, CPUs y SO.■
- Usa un formato CLARO y ORGANIZADO con emojis.■
- Separa cada máquina con una línea en blanco.■
- Mantén la información CONCISA pero COMPLETA.■
- Usa los valores EXACTOS de la información proporcionada.■
```

```
■
EJEMPLO DE FORMATO CORRECTO:■
**Resumen del Sistema MAAS**■
■
**MÁQUINA: maquinaprueba** (ID: 7mdht4)■
IP: 172.16.25.201■
RAM: 2 GB | Almacenamiento: 21.0 GB | CPUs: 1 núcleo■
SO: ubuntu jammy■
Zona: default | Pool: default■
■
**MÁQUINA: maq2** (ID: abc123)■
IP: 172.16.25.202■
RAM: 4 GB | Almacenamiento: 50.0 GB | CPUs: 2 núcleos■
SO: ubuntu focal■
Zona: default | Pool: default■
■
**Estadísticas**:■
Total: 2 máquinas■
Encendidas: 1■
Apagadas: 1■
■
Responde en español:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
        else:■
            # Pregunta general - usar Gemini para análisis contextual■
            prompt = f"""■
INFORMACIÓN ACTUAL DE LAS MÁQUINAS EN MAAS:■
{maquinas_texto}■
■
PREGUNTA DEL USUARIO: "{pregunta}"■
■
INSTRUCCIONES ESPECÍFICAS:■
- Analiza qué información es RELEVANTE para responder esta pregunta específica.■
- Responde de forma AMIGABLE y ÚTIL.■
- Usa emojis para hacer la respuesta más atractiva.■
- Si la pregunta es sobre un aspecto concreto, habla solo de ese aspecto.■
- Si es una pregunta general, da un resumen breve pero completo.■
- Si no hay información relevante, sugiere qué tipo de preguntas puedo responder.■
- Sé conversacional pero profesional.■
■
Responde en español:■
"""■
            return await generar_respuesta_gemini(prompt)■
■
    except Exception as e:■
        print(f"Error en responder_pregunta: {e}")■
        import traceback■
        traceback.print_exc()■
        return f"■ Ocurrió un error al procesar tu solicitud. Por favor, intenta de nuevo.\n\n Detalle: {str(e)}"
```

Archivo 6: ./services/maas_client.py

```
import asyncio■
from datetime import datetime■
from maas.client import connect■
import re■
import threading■
■
from config import MAAS_URL, MAAS_API_KEY■
from services.telegram_service import enviar_notificacion_telegram■
from utils.helpers import serializar_objeto_simple■
■
#=====================================================■
# Funciones básicas de MAAS■
#=====================================================■
■
async def obtener_maquinas():■
    """Obtiene lista de todas las máquinas"""■
    client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
    return await client.machines.list()■
■
async def obtener_estado_actual():■
    """Obtiene el estado actual de todas las máquinas"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        machines = await client.machines.list()■
        estados_actuales = {}■
■
        for m in machines:■
            m_full = await client.machines.get(m.system_id)■
            power_state = m_full._data.get("power_state", "unknown")■
            estados_actuales[m_full.hostname] = {■
                "power_state": power_state,■
                "system_id": m_full.system_id,■
                "ip": m_full.ip_addresses[0] if m_full.ip_addresses else "Sin IP"■
            }■
■
        return estados_actuales■
    except Exception as e:■
        print(f"Error obteniendo estados: {e}")■
        return {}■
■
async def listar_maquinas():■
    """Lista todas las máquinas en formato texto legible"""■
    client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
    machines = await client.machines.list()■
    lista_texto = ""■
    for m in machines:■
        m_full = await client.machines.get(m.system_id)■
■
        # INFORMACIÓN BÁSICA■
        hostname = m_full.hostname■
        system_id = m_full.system_id■
        status_name = m_full.status_name■
■
        # ESTADO DE ENCENDIDO■
        power_state = m_full._data.get("power_state", "unknown")■
        if power_state == "on":■
            encendido = "■ ENCENDIDA"■
        elif power_state == "off":■
            encendido = "■ APAGADA"■
        else:■
            encendido = "■ DESCONOCIDO"■
■
        # IP■
        ip_principal = m_full.ip_addresses[0] if m_full.ip_addresses else "Sin IP"■
■
        # HARDWARE■
        memoria_mb = m_full._data.get("memory", 0)■
        ram_gb = round(memoria_mb / 1024) if memoria_mb and memoria_mb > 0 else "Desconocida"■
■
        storage_mb = m_full._data.get("storage", 0)■
        storage_gb = round(storage_mb / 1024, 1) if storage_mb and storage_mb > 0 else "Desconocido"■
■
        cpu_count = m_full._data.get("cpu_count", "Desconocido")■
■
        # SISTEMA OPERATIVO■
```

```
        osystem = m_full.osystem■
        distro_series = m_full.distro_series■
■
        # ZONA Y POOL■
        zone_name = m_full.zone.name if m_full.zone else "default"■
        pool_name = m_full.pool.name if m_full.pool else "default"■
■
        lista_texto += (■
            f"■■ MÁQUINA: {hostname} ({system_id})\n"■
            f"■ Estado MAAS: {status_name}\n"■
            f"■ Estado: {encendido}\n"■
            f"■ IP: {ip_principal}\n"■
            f"■ RAM: {ram_gb} GB\n"■
            f"■ Almacenamiento: {storage_gb} GB\n"■
            f"■ CPUs: {cpu_count} núcleos\n"■
            f"■■ SO: {osystem} {distro_series}\n"■
            f"■ Zona: {zone_name} | Pool: {pool_name}\n\n"■
        )■
■
    return lista_texto■
■
async def obtener_subredes():■
    """Obtiene lista de todas las subredes"""■
    client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
    return await client.subnets.list()■
■
async def listar_subredes():■
    """Lista todas las subredes en formato texto legible"""■
    subnets = await obtener_subredes()■
    lista_texto = ""■
    for s in subnets:■
        cidr = getattr(s, "cidr", "Desconocido")■
        name = getattr(s, "name", "Sin nombre")■
        vlan = getattr(s, "vlan", "No asignada")■
        lista_texto += f"■ Subred: {name}, CIDR: {cidr}, VLAN: {vlan}\n"■
    return lista_texto■
■
#================================================■
# Control de máquinas■
#================================================■
■
async def encender_maquina(identificador):■
    """Enciende una máquina por hostname o system_id"""■
    print(f"DEBUG [encender_maquina]: Iniciando para identificador: {identificador}")■
■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquinas = await client.machines.list()■
■
        maquina_encontrada = None■
        for m in maquinas:■
            m_full = await client.machines.get(m.system_id)■
            if (m_full.hostname.lower() == identificador.lower() or m_full.system_id.lower() == identificador.lower())
                maquina_encontrada = m_full■
                break■
■
        if not maquina_encontrada:■
            print(f"DEBUG [encender_maquina]: Máquina no encontrada: {identificador}")■
            mensaje_error = f"<b>Error en comando</b>\n■ <b>Máquina no encontrada:</b> {identificador}\n■ {datetime.n
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_error)).start()■
            return f"■ No se encontró la máquina: {identificador}"■
■
        # Verificar estado actual■
        power_state = maquina_encontrada._data.get("power_state", "unknown")■
        print(f"DEBUG [encender_maquina]: Estado actual de {maquina_encontrada.hostname}: {power_state}")■
■
        if power_state == "on":■
            print(f"DEBUG [encender_maquina]: La máquina ya está encendida")■
            return f"■■ La máquina {maquina_encontrada.hostname} ya está encendida"■
■
        # Notificación de inicio de comando■
        ip_maquina = maquina_encontrada.ip_addresses[0] if maquina_encontrada.ip_addresses else "Sin IP"■
        mensaje_inicio = f"<b>Comando ejecutado</b>\n■ <b>Encendiendo:</b> {maquina_encontrada.hostname} ({ip_maquina}
        threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_inicio)).start()■
■
        print(f"DEBUG [encender_maquina]: Enviando comando power_on a MAAS")■
        # Encender la máquina■
```

```
        await maquina_encontrada.power_on()■
        await asyncio.sleep(5)■
■
        # Verificar nuevo estado■
        print(f"DEBUG [encender_maquina]: Verificando nuevo estado")■
        maquina_actualizada = await client.machines.get(maquina_encontrada.system_id)■
        nuevo_estado = maquina_actualizada._data.get("power_state", "unknown")■
■
        print(f"DEBUG [encender_maquina]: Nuevo estado: {nuevo_estado}")■
        if nuevo_estado == "on":■
            mensaje_exito = f"<b>Comando completado</b>\n■ <b>Máquina encendida:</b> {maquina_encontrada.hostname}\n■
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_exito)).start()■
            return f"■ Máquina {maquina_encontrada.hostname} encendida exitosamente"■
        else:■
            return f"■ La máquina {maquina_encontrada.hostname} se está encendiendo (puede tardar unos momentos)"■
■
    except Exception as e:■
        print(f"DEBUG [encender_maquina]: Error: {e}")■
        import traceback■
        traceback.print_exc()■
        mensaje_error = f"<b>Error en comando</b>\n■ <b>Error al encender:</b> {identificador}\n■ {str(e)}\n■ {dateti
        threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_error)).start()■
        return f"■ Error al encender la máquina: {e}"■
■
async def apagar_maquina(identificador):■
    """Apaga una máquina por hostname o system_id"""■
    print(f"DEBUG [apagar_maquina]: Iniciando para identificador: {identificador}")■
■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquinas = await client.machines.list()■
■
        maquina_encontrada = None■
        for m in maquinas:■
            m_full = await client.machines.get(m.system_id)■
            if (m_full.hostname.lower() == identificador.lower() or m_full.system_id.lower() == identificador.lower())
                maquina_encontrada = m_full■
                break■
■
        if not maquina_encontrada:■
            print(f"DEBUG [apagar_maquina]: Máquina no encontrada: {identificador}")■
            mensaje_error = f"<b>Error en comando</b>\n■ <b>Máquina no encontrada:</b> {identificador}\n■ {datetime.n
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_error)).start()■
            return f"■ No se encontró la máquina: {identificador}"■
■
        # Verificar estado actual■
        power_state = maquina_encontrada._data.get("power_state", "unknown")■
        print(f"DEBUG [apagar_maquina]: Estado actual de {maquina_encontrada.hostname}: {power_state}")■
■
        if power_state == "off":■
            print(f"DEBUG [apagar_maquina]: La máquina ya está apagada")■
            return f"■■ La máquina {maquina_encontrada.hostname} ya está apagada"■
■
        # Notificación de inicio de comando■
        ip_maquina = maquina_encontrada.ip_addresses[0] if maquina_encontrada.ip_addresses else "Sin IP"■
        mensaje_inicio = f"<b>Comando ejecutado</b>\n■ <b>Apagando:</b> {maquina_encontrada.hostname} ({ip_maquina})\n
        threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_inicio)).start()■
■
        print(f"DEBUG [apagar_maquina]: Enviando comando power_off a MAAS")■
        # Apagar la máquina■
        await maquina_encontrada.power_off()■
        await asyncio.sleep(5)■
■
        # Verificar nuevo estado■
        print(f"DEBUG [apagar_maquina]: Verificando nuevo estado")■
        maquina_actualizada = await client.machines.get(maquina_encontrada.system_id)■
        nuevo_estado = maquina_actualizada._data.get("power_state", "unknown")■
■
        print(f"DEBUG [apagar_maquina]: Nuevo estado: {nuevo_estado}")■
        if nuevo_estado == "off":■
            mensaje_exito = f"<b>Comando completado</b>\n■ <b>Máquina apagada:</b> {maquina_encontrada.hostname}\n■ {
            threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_exito)).start()■
            return f"■ Máquina {maquina_encontrada.hostname} apagada exitosamente"■
        else:■
            return f"■ La máquina {maquina_encontrada.hostname} se está apagando (puede tardar unos momentos)"■
■
    except Exception as e:■
```

```
        print(f"DEBUG [apagar_maquina]: Error: {e}")■
        import traceback■
        traceback.print_exc()■
        mensaje_error = f"<b>Error en comando</b>\n■ <b>Error al apagar:</b> {identificador}\n■ {str(e)}\n■ {datetime
        threading.Thread(target=lambda: enviar_notificacion_telegram(mensaje_error)).start()■
        return f"■ Error al apagar la máquina: {e}"■
■
async def buscar_maquina_por_ip(ip):■
    """Busca una máquina por dirección IP"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquinas = await client.machines.list()■
■
        for m in maquinas:■
            m_full = await client.machines.get(m.system_id)■
            if m_full.ip_addresses and ip in m_full.ip_addresses:■
                return m_full■
        return None■
■
    except Exception as e:■
        print(f"Error buscando máquina por IP: {e}")■
        return None■
■
async def buscar_maquina_por_nombre_o_id(identificador):■
    """Busca una máquina por nombre o system_id"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquinas = await client.machines.list()■
■
        for m in maquinas:■
            m_full = await client.machines.get(m.system_id)■
            if (m_full.hostname.lower() == identificador.lower() or■
                m_full.system_id.lower() == identificador.lower()):■
                return m_full■
        return None■
■
    except Exception as e:■
        print(f"Error buscando máquina: {e}")■
        return None■
■
#================================================■
# Funciones para Dashboard■
#================================================■
■
async def obtener_metricas_dashboard():■
    """Obtiene métricas completas para el dashboard"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        machines = await client.machines.list()■
■
        metricas = {■
            "resumen": await obtener_resumen_general(client, machines),■
            "maquinas": await obtener_detalle_maquinas(client, machines),■
            "red": await obtener_metricas_red(client),■
            "alertas": await obtener_alertas_activas(client, machines),■
            "rendimiento": await obtener_metricas_rendimiento(client, machines)■
        }■
■
        return serializar_objeto_simple(metricas)■
■
    except Exception as e:■
        print(f"Error obteniendo métricas del dashboard: {e}")■
        return {■
            "resumen": {},■
            "maquinas": [],■
            "red": {},■
            "alertas": [],■
            "rendimiento": {},■
            "error": str(e)■
        }■
■
async def obtener_resumen_general(client, machines):■
    """Obtiene resumen general del sistema"""■
    try:■
        total_maquinas = len(machines)■
        maquinas_encendidas = 0■
        maquinas_apagadas = 0■
```

```
        total_ram = 0
        total_cpu = 0
        total_almacenamiento = 0

        for m in machines:
            m_full = await client.machines.get(m.system_id)
            power_state = m_full._data.get("power_state", "unknown")

            if power_state == "on":
                maquinas_encendidas += 1
            elif power_state == "off":
                maquinas_apagadas += 1

            # Recursos
            total_ram += m_full._data.get("memory", 0)
            total_cpu += m_full._data.get("cpu_count", 0)
            total_almacenamiento += m_full._data.get("storage", 0)

        return {
            "total_maquinas": total_maquinas,
            "maquinas_encendidas": maquinas_encendidas,
            "maquinas_apagadas": maquinas_apagadas,
            "maquinas_desconocidas": total_maquinas - maquinas_encendidas - maquinas_apagadas,
            "total_ram_gb": round(total_ram / 1024, 1),
            "total_cpu_cores": total_cpu,
            "total_almacenamiento_gb": round(total_almacenamiento / 1024, 1),
            "timestamp": datetime.now().isoformat()
        }
    except Exception as e:
        print(f"Error en resumen general: {e}")
        return {}

async def obtener_detalle_maquinas(client, machines):
    """Obtiene detalle de todas las máquinas"""
    try:
        detalle_maquinas = []
        for m in machines:
            try:
                m_full = await client.machines.get(m.system_id)

                power_state = m_full._data.get("power_state", "unknown")
                status_name = m_full.status_name

                # Calcular estado de salud
                salud = "healthy"
                if status_name in ["Failed", "Error"]:
                    salud = "critical"
                elif status_name in ["Deploying", "Commissioning"]:
                    salud = "warning"
                elif power_state == "unknown":
                    salud = "unknown"

                # Extraer información de forma segura
                zona_info = "default"
                pool_info = "default"

                try:
                    if m_full.zone:
                        zona_info = getattr(m_full.zone, "name", "default")
                except:
                    pass

                try:
                    if m_full.pool:
                        pool_info = getattr(m_full.pool, "name", "default")
                except:
                    pass

                # Obtener IP de forma segura
                ip_principal = "Sin IP"
                try:
                    if m_full.ip_addresses and len(m_full.ip_addresses) > 0:
                        ip_principal = m_full.ip_addresses[0]
                except:
                    pass

                detalle_maquinas.append({
```

```
                    "hostname": m_full.hostname,■
                    "system_id": m_full.system_id,■
                    "power_state": power_state,■
                    "status": status_name,■
                    "salud": salud,■
                    "ip": ip_principal,■
                    "ram_gb": round(m_full._data.get("memory", 0) / 1024) if m_full._data.get("memory") else 0,■
                    "almacenamiento_gb": round(m_full._data.get("storage", 0) / 1024, 1) if m_full._data.get("storage"
                    "cpu_cores": m_full._data.get("cpu_count", 0),■
                    "so": f"{m_full.osystem} {m_full.distro_series}" if m_full.osystem else "NO SO",■
                    "zona": zona_info,■
                    "pool": pool_info,■
                    "ultima_actualizacion": datetime.now().isoformat()■
                })■
■
            except Exception as e:■
                print(f"Error procesando máquina {m.system_id}: {e}")■
                detalle_maquinas.append({■
                    "hostname": f"Error-{m.system_id}",■
                    "system_id": m.system_id,■
                    "power_state": "unknown",■
                    "status": "Error",■
                    "salud": "critical",■
                    "ip": "Error",■
                    "ram_gb": 0,■
                    "almacenamiento_gb": 0,■
                    "cpu_cores": 0,■
                    "so": "Error al cargar",■
                    "zona": "default",■
                    "pool": "default",■
                    "ultima_actualizacion": datetime.now().isoformat(),■
                    "error": str(e)■
                })■
■
        return detalle_maquinas■
    except Exception as e:■
        print(f"Error obteniendo detalle de máquinas: {e}")■
        return []■
■
async def obtener_metricas_red(client):■
    """Obtiene métricas de red"""■
    try:■
        subnets = await client.subnets.list()■
        metricas_red = {■
            "total_subredes": len(subnets),■
            "subredes": [],■
            "ips_utilizadas": 0,■
            "ips_disponibles": 0■
        }■
        for subnet in subnets:■
            subnet_info = {■
                "nombre": str(getattr(subnet, "name", "Sin nombre")),■
                "cidr": str(getattr(subnet, "cidr", "Desconocido")),■
                "vlan": "No asignada",■
                "space": "default",■
                "gateway": str(getattr(subnet, "gateway_ip", "No configurado"))■
            }■
            try:■
                vlan_obj = getattr(subnet, "vlan", None)■
                if vlan_obj:■
                    vlan_id = str(getattr(vlan_obj, "id", "N/A"))■
                    vlan_name = str(getattr(vlan_obj, "name", "Sin nombre"))■
                    vlan_vid = str(getattr(vlan_obj, "vid", "N/A"))■
                    subnet_info['vlan'] = f"{vlan_name} (VID: {vlan_vid}, ID: {vlan_id})"■
            except Exception as e:■
                print(f"Error procesando VLAN: {e}")■
                subnet_info['vlan'] = "Error al obtener VLAN"■
■
            try:■
                space_obj = getattr(subnet, "space", None)■
                if space_obj:■
                    space_name = str(getattr(space_obj, "name", "default"))■
                    subnet_info['space'] = space_name■
            except Exception as e:■
                print(f"Error procesando space: {e}")■
■
            metricas_red["subredes"].append(subnet_info)■
```

```
■
        return metricas_red■
    except Exception as e:■
        print(f"Error obteniendo métricas de red: {e}")■
        return {■
            "total_subredes": 0,■
            "subredes": [],■
            "ips_utilizadas": 0,■
            "ips_disponibles": 0,■
            "error": str(e)■
        }■
■
async def obtener_alertas_activas(client, machines):■
    """Identifica alertas activas en el sistema"""■
    try:■
        alertas = []■
        for m in machines:■
            m_full = await client.machines.get(m.system_id)■
            status_name = m_full.status_name■
            power_state = m_full._data.get("power_state", "unknown")■
■
            if status_name == "Failed":■
                alertas.append({■
                    "tipo": "critical",■
                    "maquina": m_full.hostname,■
                    "mensaje": "Máquina en estado Failed",■
                    "timestamp": datetime.now().isoformat()■
                })■
            elif status_name == "Error":■
                alertas.append({■
                    "tipo": "critical",■
                    "maquina": m_full.hostname,■
                    "mensaje": "Máquina en estado Error",■
                    "timestamp": datetime.now().isoformat()■
                })■
            elif power_state == "unknown":■
                alertas.append({■
                    "tipo": "warning",■
                    "maquina": m_full.hostname,■
                    "mensaje": "Estado de energía desconocido",■
                    "timestamp": datetime.now().isoformat()■
                })■
        return alertas■
    except Exception as e:■
        print(f"Error obteniendo alertas: {e}")■
        return []■
■
async def obtener_metricas_rendimiento(client, machines):■
    """Obtiene métricas de rendimiento"""■
    try:■
        maquinas_encendidas = 0■
        for m in machines:■
            m_full = await client.machines.get(m.system_id)■
            if m_full._data.get("power_state") == "on":■
                maquinas_encendidas += 1■
■
        return {■
            "uso_cpu_promedio": 0,■
            "uso_memoria_promedio": 0,■
            "io_disponible": "Normal",■
            "latencia_red": "Baja",■
            "maquinas_activas": maquinas_encendidas■
        }■
    except Exception as e:■
        print(f"Error obteniendo métricas de rendimiento: {e}")■
        return {}■
■
#================================================■
# Funciones para máquinas nuevas■
#================================================■
■
async def obtener_maquinas_nuevas():■
    """Detecta máquinas nuevas en estado 'New' que necesitan commissioning"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        machines = await client.machines.list()■
        ■
```

```
        maquinas_nuevas = []
        for m in machines:
            m_full = await client.machines.get(m.system_id)
            # Máquinas en estado 'New' son las recién detectadas
            if m_full.status_name == "New":
                ip_principal = m_full.ip_addresses[0] if m_full.ip_addresses else "Sin IP"
                maquinas_nuevas.append({
                    "hostname": m_full.hostname,
                    "system_id": m_full.system_id,
                    "ip": ip_principal,
                    "status": m_full.status_name,
                    "timestamp": datetime.now().isoformat()
                })
        
        return maquinas_nuevas
    except Exception as e:
        print(f"Error detectando máquinas nuevas: {e}")
        return []

async def abortar_commissioning(system_id):
    """Solo aborta el commissioning automático SIN configurar power"""
    try:
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)
        maquina = await client.machines.get(system_id)
        
        print(f" Abortando commissioning para: {maquina.hostname} (Estado: {maquina.status_name})")
        
        resultado_abort = ""
        
        # Solo abortar commissioning, NO configurar power
        if maquina.status_name == "Commissioning":
            try:
                await maquina.abort()
                resultado_abort = f" Commissioning abortado para {maquina.hostname}"
                print(resultado_abort)
            except Exception as abort_error:
                print(f" Error con abort(): {abort_error}")
                try:
                    await maquina.power_off()
                    resultado_abort = f" Máquina apagada para abortar commissioning: {maquina.hostname}"
                    print(resultado_abort)
                except Exception as power_error:
                    resultado_abort = f" No se pudo abortar commissioning: {str(abort_error)}"
                    print(resultado_abort)
        elif maquina.status_name == "New":
            resultado_abort = f"  Máquina {maquina.hostname} en estado New"
            print(resultado_abort)
        else:
            resultado_abort = f"  La máquina {maquina.hostname} no está en commissioning. Estado: {maquina.status_nam
            print(resultado_abort)
        
        return resultado_abort
            
    except Exception as e:
        print(f" Error general al abortar commissioning: {e}")
        return f" Error al abortar commissioning: {str(e)}"


async def configurar_power_virsh(system_id, vm_id):
    """Configura power Virsh y cambia el hostname al VM ID en MAAS 3.5.8"""
    try:
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)
        maquina = await client.machines.get(system_id)
        
        hostname_original = maquina.hostname
        print(f" Configurando power Virsh para: {hostname_original} -> VM ID: {vm_id}")
        
        # Parámetros para Virsh
        power_params = {
            "power_address": "qemu+ssh://branvictus@172.16.25.1/system",
            "power_id": vm_id
        }
        
        print(f" Estado actual - Power type: {maquina.power_type}")
        print(f" Estado actual - Hostname: {hostname_original}")
        
        resultados = []
```

```
        ■
        # PASO 1: Configurar power Virsh■
        try:■
            await maquina.set_power(■
                power_type="virsh",■
                power_parameters=power_params■
            )■
            resultados.append(f"■ Power Virsh configurado con VM ID '{vm_id}'")■
            print(f"■ set_power ejecutado exitosamente")■
        except Exception as e_power:■
            resultados.append(f"■ Error configurando power: {str(e_power)}")■
            print(f"■ Error en set_power: {e_power}")■
        ■
        # PASO 2: Cambiar hostname al VM ID■
        try:■
            resultado_hostname = await cambiar_hostname_maas(system_id, vm_id)■
            resultados.append(resultado_hostname)■
        except Exception as e_hostname:■
            resultados.append(f"■ Error cambiando hostname: {str(e_hostname)}")■
            print(f"■ Error cambiando hostname: {e_hostname}")■
        ■
        # Verificar configuración final■
        maquina_actualizada = await client.machines.get(system_id)■
        print(f"■ Configuración final - Power type: {maquina_actualizada.power_type}")■
        print(f"■ Configuración final - Hostname: {maquina_actualizada.hostname}")■
        ■
        # Combinar resultados■
        if any("■" in resultado for resultado in resultados):■
            mensaje_final = "■■ Configuración parcial:\n" + "\n".join(resultados)■
        else:■
            mensaje_final = "■ Configuración completada:\n" + "\n".join(resultados)■
        ■
        return mensaje_final■
            ■
    except Exception as e:■
        print(f"■ Error en configurar_power_virsh: {e}")■
        import traceback■
        traceback.print_exc()■
        return f"■ Error en configuración: {str(e)}"■
■
async def debug_maquina_detallado(system_id):■
    """Debug detallado del objeto Machine en MAAS 3.5.8"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquina = await client.machines.get(system_id)■
        ■
        print("=== DEBUG DETALLADO MAAS 3.5.8 ===")■
        print(f"Hostname: {maquina.hostname}")■
        print(f"System ID: {maquina.system_id}")■
        print(f"Status: {maquina.status_name}")■
        ■
        # Atributos comunes de power■
        power_attrs = ['power_type', 'power_parameters', 'power_state']■
        for attr in power_attrs:■
            if hasattr(maquina, attr):■
                value = getattr(maquina, attr)■
                print(f"{attr}: {value}")■
            else:■
                print(f"{attr}: NO EXISTE")■
        ■
        # Métodos disponibles■
        methods = [m for m in dir(maquina) if not m.startswith('_') and callable(getattr(maquina, m))]■
        print(f"Métodos disponibles: {methods}")■
        ■
        # Métodos específicos de power■
        power_methods = [m for m in methods if 'power' in m.lower()]■
        print(f"Métodos de power: {power_methods}")■
        ■
        # Métodos de guardado/actualización■
        save_methods = [m for m in methods if any(x in m.lower() for x in ['save', 'update', 'edit', 'configure'])]■
        print(f"Métodos de guardado: {save_methods}")■
        ■
        return {■
            "hostname": maquina.hostname,■
            "power_attrs": {attr: getattr(maquina, attr, "NO EXISTE") for attr in power_attrs},■
            "power_methods": power_methods,■
            "save_methods": save_methods■
```

```
        }
    except Exception as e:
        return {"error": str(e)}

async def listar_todas_maquinas_con_ids():
    """Lista todas las máquinas con sus system_ids para debugging"""
    try:
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)
        machines = await client.machines.list()

        maquinas_info = []
        for m in machines:
            m_full = await client.machines.get(m.system_id)
            maquinas_info.append({
                "hostname": m_full.hostname,
                "system_id": m_full.system_id,
                "status": m_full.status_name,
                "power_type": m_full.power_type,
                "ip": m_full.ip_addresses[0] if m_full.ip_addresses else "Sin IP"
            })

        return maquinas_info
    except Exception as e:
        print(f"Error listando máquinas: {e}")
        return []

async def debug_power_parameters(system_id):
    """Debug específico para parámetros de power"""
    try:
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)
        maquina = await client.machines.get(system_id)

        print("=== DEBUG POWER PARAMETERS ===")
        print(f"Hostname: {maquina.hostname}")
        print(f"Power type: {maquina.power_type}")
        print(f"Power state: {maquina.power_state}")

        # Obtener parámetros actuales
        try:
            params = await maquina.get_power_parameters()
            print(f"Power parameters: {params}")
        except Exception as e:
            print(f"Error obteniendo power parameters: {e}")

        return {
            "hostname": maquina.hostname,
            "power_type": maquina.power_type,
            "power_state": str(maquina.power_state),
            "power_parameters": params if 'params' in locals() else f"Error: {e}"
        }
    except Exception as e:
        return {"error": str(e)}

async def cambiar_hostname_maas(system_id, nuevo_hostname):
    """Cambia el hostname de una máquina en MAAS"""
    try:
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)
        maquina = await client.machines.get(system_id)

        print(f"■■ Cambiando hostname de '{maquina.hostname}' a '{nuevo_hostname}'")

        # En MAAS 3.5.8, podemos intentar varias formas de cambiar el hostname

        # Método 1: Asignación directa al atributo hostname + save()
        try:
            maquina.hostname = nuevo_hostname
            await maquina.save()
            print(f"■ Hostname cambiado exitosamente a '{nuevo_hostname}'")
            return f"■ Hostname cambiado a '{nuevo_hostname}'"
        except Exception as e1:
            print(f"■ Método 1 falló: {e1}")

            # Método 2: Usar update si existe (aunque en el debug no vimos update, por si acaso)
            try:
                if hasattr(maquina, 'update'):
                    await maquina.update(hostname=nuevo_hostname)
                    print(f"■ Hostname cambiado usando update()")
```

```
                    return f"■ Hostname cambiado a '{nuevo_hostname}'"■
                except Exception as e2:■
                    print(f"■ Método 2 falló: {e2}")■
                    ■
                    # Método 3: API REST directa■
                    try:■
                        return await cambiar_hostname_api_directa(system_id, nuevo_hostname)■
                    except Exception as e3:■
                        print(f"■ Método 3 falló: {e3}")■
                        raise Exception(f"Todos los métodos fallaron: {e1}, {e2}, {e3}")■
                        ■
    except Exception as e:■
        print(f"■ Error cambiando hostname: {e}")■
        return f"■ Error cambiando hostname: {str(e)}"■
■
async def cambiar_hostname_api_directa(system_id, nuevo_hostname):■
    """Cambia el hostname usando la API REST directamente"""■
    try:■
        import aiohttp■
        import json■
        ■
        # URL de la API de MAAS■
        url = f"{MAAS_URL}/api/2.0/machines/{system_id}/"■
        ■
        # Headers con autenticación■
        headers = {■
            'Authorization': f'ApiKey {MAAS_API_KEY}',■
            'Content-Type': 'application/x-www-form-urlencoded'■
        }■
        ■
        # Datos para cambiar hostname■
        form_data = aiohttp.FormData()■
        form_data.add_field('hostname', nuevo_hostname)■
        ■
        # Hacer la petición PUT■
        async with aiohttp.ClientSession() as session:■
            async with session.put(url, headers=headers, data=form_data) as response:■
                if response.status == 200:■
                    return f"■ Hostname cambiado a '{nuevo_hostname}'"■
                else:■
                    error_text = await response.text()■
                    return f"■ Error en API: {response.status} - {error_text}"■
                    ■
    except Exception as e:■
        return f"■ Error cambiando hostname via API: {str(e)}"■
■
# Agregar estas funciones al archivo existente■
■
async def listar_maquinas_para_commissioning():■
    """Lista solo máquinas disponibles para commissioning (excluye deployed)"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        machines = await client.machines.list()■
        ■
        maquinas_disponibles = []■
        for maquina in machines:■
            estado = getattr(maquina, 'status_name', f'Código: {maquina.status}')■
            # Solo incluir máquinas que NO están en estado Deployed■
            if estado != 'Deployed':■
                ip_principal = maquina.ip_addresses[0] if maquina.ip_addresses else "Sin IP"■
                maquinas_disponibles.append({■
                    'hostname': maquina.hostname,■
                    'system_id': maquina.system_id,■
                    'status': estado,■
                    'ip': ip_principal■
                })■
        ■
        return maquinas_disponibles■
    except Exception as e:■
        print(f"Error listando máquinas para commissioning: {e}")■
        return []■
■
async def ejecutar_commissioning(system_id, opciones=None):■
    """Ejecuta commissioning en una máquina específica"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquina = await client.machines.get(system_id)■
```

```
        ■
        # Configuración por defecto - SIN scripts específicos (usa los por defecto)■
        opciones_default = {■
            'enable_ssh': True,■
            'skip_networking': False,■
            'skip_storage': False,■
            # No especificamos commissioning_scripts para usar los por defecto de MAAS■
        }■
        ■
        if opciones:■
            opciones_default.update(opciones)■
        ■
        # Verificar estado adecuado■
        estado_actual = getattr(maquina, 'status_name', 'Desconocido')■
        estados_validos = ['Ready', 'New', 'Failed commissioning']■
        ■
        if estado_actual not in estados_validos:■
            return {■
                'success': False,■
                'message': f'■ La máquina no está en estado adecuado para commissioning. Estado actual: {estado_actual
                'estado_actual': estado_actual■
            }■
        ■
        # Ejecutar commissioning■
        print(f"■ Ejecutando commissioning en {maquina.hostname} con opciones: {opciones_default}")■
        resultado = await maquina.commission(**opciones_default)■
        ■
        return {■
            'success': True,■
            'message': f'■ Commissioning iniciado para {maquina.hostname}',■
            'hostname': maquina.hostname,■
            'system_id': system_id■
        }■
        ■
    except Exception as e:■
        print(f"■ Error en commissioning: {e}")■
        return {■
            'success': False,■
            'message': f'■ Error al iniciar commissioning: {str(e)}'■
        }■
async def obtener_estado_commissioning(system_id):■
    """Obtiene el estado actual del commissioning de una máquina"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquina = await client.machines.get(system_id)■
        ■
        return {■
            'hostname': maquina.hostname,■
            'system_id': system_id,■
            'status': getattr(maquina, 'status_name', 'Desconocido'),■
            'status_code': maquina.status■
        }■
    except Exception as e:■
        return {■
            'error': str(e)■
        }■
■
async def listar_maquinas_para_deploy():■
    """Lista solo máquinas disponibles para deploy (estado Ready)"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        machines = await client.machines.list()■
        maquinas_lista = []■
        for maquina in machines:■
            estado = getattr(maquina, 'status_name', f'Código: {maquina.status}')■
            # Solo incluir máquinas en estado Ready para deploy■
            if estado == 'Ready':■
                ip_principal = maquina.ip_addresses[0] if maquina.ip_addresses else "Sin IP"■
                maquinas_lista.append({■
                    'hostname': maquina.hostname,■
                    'system_id': maquina.system_id,■
                    'status': estado,■
                    'ip': ip_principal,■
                    'osystem': getattr(maquina, 'osystem', 'No definido'),■
                    'architecture': getattr(maquina, 'architecture', 'No definida'),■
                    'memory_gb': round(getattr(maquina, 'memory', 0) / 1024, 1),■
                    'cpu_count': getattr(maquina, 'cpu_count', 'N/A')■
```

```
                })■
        return maquinas_lista■
    except Exception as e:■
        print(f"Error listando máquinas para deploy: {e}")■
        return []■
■
async def ejecutar_deploy(system_id, opciones=None):■
    """Ejecuta deploy en una máquina específica"""■
    try:■
        client = await connect(MAAS_URL, apikey=MAAS_API_KEY)■
        maquina = await client.machines.get(system_id)■
■
        # Verificar estado■
        estado_actual = getattr(maquina, 'status_name', 'Desconocido')■
        if estado_actual != 'Ready':■
            return {■
                'success': False,■
                'message': f'■ La máquina no está en estado Ready. Estado actual: {estado_actual}'■
            }■
■
        # Preparar parámetros básicos■
        deploy_params = {■
            'wait': False■
        }■
■
        # Agregar parámetros opcionales si se proporcionan■
        if opciones:■
            if 'user_data' in opciones:■
                deploy_params['user_data'] = opciones['user_data']■
            if 'distro_series' in opciones:■
                deploy_params['distro_series'] = opciones['distro_series']■
            if 'hwe_kernel' in opciones:■
                deploy_params['hwe_kernel'] = opciones['hwe_kernel']■
■
        # Ejecutar deploy con los parámetros disponibles■
        resultado = await maquina.deploy(**deploy_params)■
■
        return {■
            'success': True,■
            'message': f'■ Deploy iniciado para {maquina.hostname}',■
            'hostname': maquina.hostname,■
            'system_id': system_id■
        }■
■
    except Exception as e:■
        print(f"Error en deploy: {e}")■
        return {■
            'success': False,■
            'message': f'■ Error al iniciar deploy: {str(e)}'■
        }
```

Archivo 7: ./services/gemini_service.py

```
import google.generativeai as genai■
from config import GEMINI_API_KEY■
■
genai.configure(api_key=GEMINI_API_KEY)■
model = genai.GenerativeModel('gemini-2.0-flash')■
■
async def generar_respuesta_gemini(prompt: str) -> str:■
    """Versión asíncrona de la función Gemini"""■
    try:■
        respuesta = model.generate_content(prompt)■
        return respuesta.text.strip()■
    except Exception as e:■
        return f"Error al generar respuesta: {e}"
```

Archivo 8: ./services/telegram_service.py

```
import requests■
from config import TELEGRAM_BOT_TOKEN, TELEGRAM_CHAT_ID■
■
def enviar_notificacion_telegram(mensaje):■
    """Envía notificación a Telegram (para el servicio de monitoreo)"""■
    try:■
        if not TELEGRAM_BOT_TOKEN or not TELEGRAM_CHAT_ID:■
            print("Configuración de Telegram no completada")■
            return False■
        ■
        url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"■
        payload = {■
            "chat_id": TELEGRAM_CHAT_ID,■
            "text": mensaje,■
            "parse_mode": "HTML"■
        }■
        ■
        response = requests.post(url, json=payload, timeout=10)■
        return response.status_code == 200■
    except Exception as e:■
        print(f"Error enviando notificación a Telegram: {e}")■
        return False■
■
def enviar_mensaje_telegram(chat_id, mensaje, parse_mode='HTML'):■
    """Envía mensaje a un chat específico de Telegram"""■
    try:■
        if not TELEGRAM_BOT_TOKEN:■
            print("Token de Telegram no configurado")■
            return False■
        ■
        url = f"https://api.telegram.org/bot{TELEGRAM_BOT_TOKEN}/sendMessage"■
        payload = {■
            "chat_id": chat_id,■
            "text": mensaje,■
            "parse_mode": parse_mode■
        }■
        ■
        response = requests.post(url, json=payload, timeout=10)■
        return response.status_code == 200■
    except Exception as e:■
        print(f"Error enviando mensaje a Telegram: {e}")■
        return False
```