Guía Paso a Paso: Asistente Meteorológico para Rutas

Tabla de Contenidos

1. Obtención de API Keys

1.1 Telegram Bot Token

1. **Abrir Telegram** en cualquier dispositivo
2. **Buscar @BotFather** en la barra de búsqueda
3. **Iniciar conversación** presionando "START"
4. **Crear nuevo bot** escribiendo: `/newbot`
5. **Seguir instrucciones**:
   - Nombre del bot: `Asistente Meteorológico`
   - Username: `weather_route_assistant_bot` (debe terminar en "bot")
6. **Copiar el token** que te proporciona BotFather
   - Formato: `123456789:ABCdefGHIjklMNOpqrSTUvwxYZ`

1.2 meteoblue Free API

1. **Ir a**: https://www.meteoblue.com/
2. **Registrarse** gratis en el sitio web
3. **Confirmar uso no comercial** en el email de activación
4. **Acceder al dashboard** y navegar a "API Keys"
5. **Activar Free Weather API** (5,000 llamadas/año)
6. **Copiar API key** generada

1.3 AEMET OpenData

1. **Ir a**: https://opendata.aemet.es/centrodedescargas/inicio
2. **Hacer clic** en "Obtención de API Key" → "Solicitar"
3. **Introducir email** y completar captcha
4. **Confirmar** en el email recibido
5. **Recibir API key** en segundo email (válida indefinidamente)
6. **Copiar la clave** (formato JWT muy largo)

1.4 OpenRouteService

1. **Ir a**: https://openrouteservice.org/
2. **Registrarse** haciendo clic en "Sign Up"
3. **Confirmar email** haciendo clic en el enlace de activación
4. **Iniciar sesión** en el dashboard
5. **Ir a "TOKENS"** en la parte inferior
6. **Crear nuevo token**:
   - Name: `weather-assistant`
   - Type: `Free` (40 requests/min, 2000 requests/day)
7. **Copiar el token** generado

1.5 Windy API (Opcional - de pago)

1. **Ir a**: https://api.windy.com/point-forecast
2. **Registrarse** para obtener API key
3. **Nota**: Es de pago (~990€/año profesional)
4. **Para desarrollo**: usar datos de prueba o skip esta API

1.6 Meteored/tiempo.com

1. **Ir a**: https://www.tiempo.com/api/
2. **Registrarse** completando todos los campos
3. **Confirmar email** haciendo clic en enlace
4. **Acceder al panel** con email y contraseña
5. **Obtener URLs de API** desde "Una localidad"
6. **Importante**: Requiere enlace visible a tiempo.com en tu web

2. Configuración del Entorno

2.1 Instalación de Software Base

```
# Actualizar sistema (Ubuntu/Debian)
sudo apt update && sudo apt upgrade -y

# Instalar Node.js (versión 18+)
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

# Verificar instalación
node --version
npm --version

# Instalar Docker y Docker Compose
curl -fsSL https://get.docker.com -o get-docker.sh
sudo sh get-docker.sh
sudo usermod -aG docker $USER

# Instalar Docker Compose
sudo apt-get install docker-compose-plugin

# Instalar Redis (para desarrollo local)
sudo apt-get install redis-server
```

2.2 Configuración de VS Code

1. **Instalar VS Code**: https://code.visualstudio.com/
2. **Instalar extensiones esenciales**:
   - JavaScript (ES6) code snippets
   - Docker
   - GitLens
   - Thunder Client (para pruebas API)
   - ENV (para archivos .env)
   - Auto Rename Tag
   - Bracket Pair Colorizer 2

2.3 Configuración de Git

```
# Configurar Git
git config --global user.name "Tu Nombre"
git config --global user.email "tu-email@example.com"

# Crear repositorio
mkdir weather-route-assistant
```

```
cd weather-route-assistant
git init
```

3. Estructura del Proyecto

3.1 Crear Estructura de Carpetas

```
# Ejecutar desde la raíz del proyecto
mkdir -p services/{telegram-bot,route-processor,weather-orchestrator,response-composer}/{
mkdir -p services/telegram-bot/src/{handlers,middleware,utils}
mkdir -p services/route-processor/src/{parsers,apis}
mkdir -p services/weather-orchestrator/src/{providers,cache}
mkdir -p services/response-composer/src/{templates,formatters}
mkdir -p config/{nginx,redis}
mkdir -p scripts
mkdir -p docs
```

3.2 Estructura Final

```
weather-route-assistant/
├── .env.example
├── .gitignore
├── docker-compose.yml
├── README.md
├── package.json
├── services/
│   ├── telegram-bot/
│   │   ├── Dockerfile
│   │   ├── package.json
│   │   ├── src/
│   │   │   ├── index.js
│   │   │   ├── handlers/
│   │   │   │   ├── messageHandler.js
│   │   │   │   └── fileHandler.js
│   │   │   ├── middleware/
│   │   │   │   ├── rateLimiter.js
│   │   │   │   └── auth.js
│   │   │   └── utils/
│   │   │       ├── logger.js
│   │   │       └── helpers.js
│   │   └── tests/
│   ├── route-processor/
│   │   ├── Dockerfile
│   │   ├── package.json
│   │   ├── src/
│   │   │   ├── index.js
│   │   │   ├── parsers/
│   │   │   │   └── gpxParser.js
│   │   │   └── apis/
│   │   │       ├── osrmClient.js
│   │   │       └── orsClient.js
```

```
│   │   └── tests/
│   ├── weather-orchestrator/
│   │   ├── Dockerfile
│   │   ├── package.json
│   │   ├── src/
│   │   │   ├── index.js
│   │   │   ├── providers/
│   │   │   │   ├── meteoblue.js
│   │   │   │   ├── aemet.js
│   │   │   │   ├── windy.js
│   │   │   │   └── meteored.js
│   │   │   └── cache/
│   │   │       ├── redisClient.js
│   │   │       └── rateLimiter.js
│   │   └── tests/
│   └── response-composer/
│       ├── Dockerfile
│       ├── package.json
│       ├── src/
│       │   ├── index.js
│       │   ├── templates/
│       │   │   └── briefingTemplate.js
│       │   └── formatters/
│       │       └── telegramFormatter.js
│       └── tests/
├── config/
│   ├── nginx/
│   │   └── nginx.conf
│   └── redis/
│       └── redis.conf
└── scripts/
    ├── setup.sh
    ├── deploy.sh
    └── test.sh
```

4. Configuración de Dependencias

4.1 package.json Principal

```
{
  "name": "weather-route-assistant",
  "version": "1.0.0",
  "description": "Asistente meteorológico para rutas de senderismo",
  "main": "index.js",
  "scripts": {
    "dev": "docker-compose up --build",
    "prod": "docker-compose -f docker-compose.yml -f docker-compose.prod.yml up -d",
    "test": "npm run test --workspaces",
    "setup": "bash scripts/setup.sh"
  },
  "workspaces": [
    "services/*"
  ],
```

```
  "keywords": ["weather", "routing", "telegram", "hiking"],
  "author": "Tu Nombre",
  "license": "MIT",
  "devDependencies": {
    "nodemon": "^3.0.1",
    "jest": "^29.7.0"
  }
}
```

4.2 Telegram Bot Dependencies

```
{
  "name": "telegram-bot",
  "version": "1.0.0",
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js",
    "test": "jest"
  },
  "dependencies": {
    "telegraf": "^4.15.6",
    "axios": "^1.6.2",
    "ioredis": "^5.3.2",
    "winston": "^3.11.0",
    "dotenv": "^16.3.1",
    "express": "^4.18.2",
    "multer": "^1.4.5-lts.1",
    "helmet": "^7.1.0"
  }
}
```

4.3 Route Processor Dependencies

```
{
  "name": "route-processor",
  "version": "1.0.0",
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js"
  },
  "dependencies": {
    "@we-gold/gpxjs": "^2.0.0",
    "axios": "^1.6.2",
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "helmet": "^7.1.0",
    "winston": "^3.11.0",
    "dotenv": "^16.3.1"
  }
}
```

4.4 Weather Orchestrator Dependencies

```
{
  "name": "weather-orchestrator",
  "version": "1.0.0",
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "axios": "^1.6.2",
    "ioredis": "^5.3.2",
    "node-cron": "^3.0.3",
    "winston": "^3.11.0",
    "dotenv": "^16.3.1",
    "helmet": "^7.1.0",
    "cors": "^2.8.5"
  }
}
```

4.5 Response Composer Dependencies

```
{
  "name": "response-composer",
  "version": "1.0.0",
  "scripts": {
    "start": "node src/index.js",
    "dev": "nodemon src/index.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "winston": "^3.11.0",
    "dotenv": "^16.3.1",
    "helmet": "^7.1.0",
    "cors": "^2.8.5"
  }
}
```

5. Código Completo

5.1 Variables de Entorno (.env.example)

```
# Telegram Bot
BOT_TOKEN=tu_bot_token_aqui
WEBHOOK_URL=https://tu-dominio.com/webhook

# APIs Meteorológicas
METEOBLUE_API_KEY=tu_meteoblue_api_key
AEMET_API_KEY=tu_aemet_api_key
```

```
WINDY_API_KEY=tu_windy_api_key_opcional
METEORED_API_KEY=tu_meteored_api_key

# Routing APIs
ORS_API_KEY=tu_openrouteservice_api_key
OSRM_URL=http://router.project-osrm.org

# Redis
REDIS_URL=redis://redis:6379

# URLs Internas
ROUTE_PROCESSOR_URL=http://route-processor:3001
WEATHER_ORCHESTRATOR_URL=http://weather-orchestrator:3002
RESPONSE_COMPOSER_URL=http://response-composer:3003

# Configuración
NODE_ENV=development
PORT=3000
LOG_LEVEL=info

# Rate Limiting
RATE_LIMIT_WINDOW_MS=60000
RATE_LIMIT_MAX_REQUESTS=30
```

5.2 Docker Compose

```
version: '3.8'

services:
  redis:
    image: redis:7-alpine
    container_name: weather-redis
    ports:
      - "6379:6379"
    volumes:
      - redis_data:/data
      - ./config/redis/redis.conf:/usr/local/etc/redis/redis.conf
    command: redis-server /usr/local/etc/redis/redis.conf
    restart: unless-stopped

  telegram-bot:
    build: ./services/telegram-bot
    container_name: weather-telegram-bot
    depends_on:
      - redis
      - route-processor
      - weather-orchestrator
      - response-composer
    environment:
      - BOT_TOKEN=${BOT_TOKEN}
      - WEBHOOK_URL=${WEBHOOK_URL}
      - REDIS_URL=redis://redis:6379
      - ROUTE_PROCESSOR_URL=http://route-processor:3001
      - WEATHER_ORCHESTRATOR_URL=http://weather-orchestrator:3002
      - RESPONSE_COMPOSER_URL=http://response-composer:3003
```

```
      - NODE_ENV=${NODE_ENV:-development}
    ports:
      - "3000:3000"
    volumes:
      - ./services/telegram-bot/logs:/app/logs
    restart: unless-stopped

  route-processor:
    build: ./services/route-processor
    container_name: weather-route-processor
    depends_on:
      - redis
    environment:
      - REDIS_URL=redis://redis:6379
      - ORS_API_KEY=${ORS_API_KEY}
      - OSRM_URL=${OSRM_URL}
      - NODE_ENV=${NODE_ENV:-development}
    ports:
      - "3001:3001"
    volumes:
      - ./services/route-processor/logs:/app/logs
    restart: unless-stopped

  weather-orchestrator:
    build: ./services/weather-orchestrator
    container_name: weather-orchestrator
    depends_on:
      - redis
    environment:
      - REDIS_URL=redis://redis:6379
      - METEOBLUE_API_KEY=${METEOBLUE_API_KEY}
      - AEMET_API_KEY=${AEMET_API_KEY}
      - WINDY_API_KEY=${WINDY_API_KEY}
      - METEORED_API_KEY=${METEORED_API_KEY}
      - NODE_ENV=${NODE_ENV:-development}
    ports:
      - "3002:3002"
    volumes:
      - ./services/weather-orchestrator/logs:/app/logs
    restart: unless-stopped

  response-composer:
    build: ./services/response-composer
    container_name: weather-response-composer
    environment:
      - NODE_ENV=${NODE_ENV:-development}
    ports:
      - "3003:3003"
    volumes:
      - ./services/response-composer/logs:/app/logs
    restart: unless-stopped

volumes:
  redis_data:
```

5.3 Telegram Bot Principal

```
// services/telegram-bot/src/index.js
const { Telegraf } = require('telegraf');
const express = require('express');
const helmet = require('helmet');
const winston = require('winston');
const Redis = require('ioredis');
require('dotenv').config();

const messageHandler = require('./handlers/messageHandler');
const fileHandler = require('./handlers/fileHandler');
const rateLimiter = require('./middleware/rateLimiter');
const logger = require('./utils/logger');

class WeatherBot {
  constructor() {
    this.bot = new Telegraf(process.env.BOT_TOKEN);
    this.app = express();
    this.redis = new Redis(process.env.REDIS_URL);
    this.setupMiddleware();
    this.setupHandlers();
    this.setupWebhook();
  }

  setupMiddleware() {
    this.app.use(helmet());
    this.app.use(express.json({ limit: '10mb' }));
    this.app.use(rateLimiter);

    // Middleware para logging de requests
    this.app.use((req, res, next) => {
      logger.info(`${req.method} ${req.path}`, {
        ip: req.ip,
        userAgent: req.get('User-Agent')
      });
      next();
    });
  }

  setupHandlers() {
    // Handler para comando start
    this.bot.start((ctx) => {
      const welcomeMessage = `
 ¡Hola! Soy tu asistente meteorológico para rutas.

 **¿Qué puedo hacer?**
• Analizar rutas descritas en texto
• Procesar archivos GPX
• Darte briefings meteorológicos detallados
• Alertarte sobre riesgos climáticos

 **Ejemplos de uso:**
"Voy a hacer una ruta por La Pedriza y pasar por el Yelmo, ¿qué tiempo habrá?"
"Ruta circular por Siete Picos saliendo mañana a las 8:00"
```

```
 **También puedes enviarme un archivo GPX** y te daré el pronóstico para toda la ruta.

¡Prueba describiendo tu próxima aventura! 
      `;

      ctx.replyWithHTML(welcomeMessage);
    });

    // Handler para mensajes de texto
    this.bot.on('text', async (ctx) => {
      try {
        await messageHandler.handleTextMessage(ctx, this.redis);
      } catch (error) {
        logger.error('Error handling text message:', error);
        await ctx.reply('⚠ Error procesando tu mensaje. Inténtalo de nuevo en unos momen
      }
    });

    // Handler para archivos
    this.bot.on('document', async (ctx) => {
      try {
        await fileHandler.handleFileUpload(ctx, this.redis);
      } catch (error) {
        logger.error('Error handling file:', error);
        await ctx.reply('⚠ Error procesando el archivo. Asegúrate de que sea un GPX váli
      }
    });

    // Handler para errores
    this.bot.catch((err, ctx) => {
      logger.error('Bot error:', err);
      ctx.reply('⚠ Ocurrió un error inesperado. El equipo técnico ha sido notificado.');
    });
  }

  setupWebhook() {
    this.app.use(this.bot.webhookCallback('/webhook'));

    // Health check endpoint
    this.app.get('/health', (req, res) => {
      res.json({
        status: 'OK',
        timestamp: new Date().toISOString(),
        uptime: process.uptime()
      });
    });

    // Webhook info endpoint
    this.app.get('/webhook-info', async (req, res) => {
      try {
        const webhookInfo = await this.bot.telegram.getWebhookInfo();
        res.json(webhookInfo);
      } catch (error) {
        res.status(500).json({ error: error.message });
      }
    });
```

```
  }

  async start() {
    const port = process.env.PORT || 3000;

    // Configurar webhook si está en producción
    if (process.env.NODE_ENV === 'production') {
      await this.bot.telegram.setWebhook(`${process.env.WEBHOOK_URL}/webhook`);
      logger.info('Webhook configurado:', process.env.WEBHOOK_URL);
    } else {
      // Usar polling en desarrollo
      this.bot.launch();
      logger.info('Bot iniciado en modo polling para desarrollo');
    }

    this.app.listen(port, () => {
      logger.info(`Servidor iniciado en puerto ${port}`);
    });

    // Graceful shutdown
    process.once('SIGINT', () => this.bot.stop('SIGINT'));
    process.once('SIGTERM', () => this.bot.stop('SIGTERM'));
  }
}

const bot = new WeatherBot();
bot.start().catch(error => {
  logger.error('Error starting bot:', error);
  process.exit(1);
});
```

5.4 Message Handler

```
// services/telegram-bot/src/handlers/messageHandler.js
const axios = require('axios');
const logger = require('../utils/logger');

class MessageHandler {
  async handleTextMessage(ctx, redis) {
    const userId = ctx.from.id;
    const messageText = ctx.message.text;
    const messageId = await ctx.reply('🔍 Procesando tu consulta meteorológica...');

    try {
      // Validar entrada
      if (messageText.length < 10) {
        await ctx.telegram.editMessageText(
          ctx.chat.id,
          messageId.message_id,
          undefined,
          '❌ Por favor, describe tu ruta con más detalle.\n\nEjemplo: "Ruta por La Pedri
        );
        return;
      }
```

```
      logger.info('Processing route description:', {
        userId,
        messageLength: messageText.length,
        preview: messageText.substring(0, 100)
      });

      // 1. Procesar descripción de ruta
      const routeData = await this.processRouteDescription(messageText, userId);

      await ctx.telegram.editMessageText(
        ctx.chat.id,
        messageId.message_id,
        undefined,
        '🗺️ Ruta procesada, obteniendo datos meteorológicos...'
      );

      // 2. Obtener datos meteorológicos
      const weatherData = await this.getWeatherForRoute(routeData);

      await ctx.telegram.editMessageText(
        ctx.chat.id,
        messageId.message_id,
        undefined,
        '📊 Componiendo briefing meteorológico...'
      );

      // 3. Componer respuesta
      const briefing = await this.composeBriefing(weatherData, routeData);

      // 4. Enviar respuesta final
      await ctx.telegram.deleteMessage(ctx.chat.id, messageId.message_id);
      await ctx.replyWithHTML(briefing.text, briefing.keyboard);

      // Log successful processing
      logger.info('Successfully processed route request:', {
        userId,
        routePoints: routeData.waypoints?.length || 0,
        weatherSources: weatherData.sources?.length || 0
      });

    } catch (error) {
      logger.error('Error processing text message:', error);

      await ctx.telegram.editMessageText(
        ctx.chat.id,
        messageId.message_id,
        undefined,
        '⚠️ Error procesando la consulta. Posibles causas:\n' +
        '• La descripción no es suficientemente específica\n' +
        '• Problema temporal con las APIs meteorológicas\n' +
        '• Ubicación no encontrada\n\n' +
        'Inténtalo de nuevo con una descripción más detallada.'
      );
    }
  }
```

```
  async processRouteDescription(description, userId) {
    try {
      const response = await axios.post(
        `${process.env.ROUTE_PROCESSOR_URL}/process-description`,
        {
          description,
          userId,
          timestamp: new Date().toISOString()
        },
        { timeout: 30000 }
      );

      return response.data;
    } catch (error) {
      if (error.code === 'ECONNABORTED') {
        throw new Error('Timeout procesando la ruta');
      }
      throw new Error(`Error en servicio de rutas: ${error.message}`);
    }
  }

  async getWeatherForRoute(routeData) {
    try {
      const response = await axios.post(
        `${process.env.WEATHER_ORCHESTRATOR_URL}/weather-for-route`,
        routeData,
        { timeout: 45000 }
      );

      return response.data;
    } catch (error) {
      if (error.code === 'ECONNABORTED') {
        throw new Error('Timeout obteniendo datos meteorológicos');
      }
      throw new Error(`Error en servicio meteorológico: ${error.message}`);
    }
  }

  async composeBriefing(weatherData, routeData) {
    try {
      const response = await axios.post(
        `${process.env.RESPONSE_COMPOSER_URL}/compose-briefing`,
        {
          weatherData,
          routeData,
          timestamp: new Date().toISOString()
        },
        { timeout: 15000 }
      );

      return response.data;
    } catch (error) {
      throw new Error(`Error componiendo respuesta: ${error.message}`);
    }
  }
}
```

```
module.exports = new MessageHandler();
```

5.5 Rate Limiter Middleware

```
// services/telegram-bot/src/middleware/rateLimiter.js
const Redis = require('ioredis');
const redis = new Redis(process.env.REDIS_URL);

const rateLimiter = async (req, res, next) => {
  try {
    const userId = req.body?.from?.id || req.ip;
    const key = `rate_limit:${userId}`;

    const windowMs = parseInt(process.env.RATE_LIMIT_WINDOW_MS) || 60000;
    const maxRequests = parseInt(process.env.RATE_LIMIT_MAX_REQUESTS) || 30;

    const current = await redis.incr(key);

    if (current === 1) {
      await redis.expire(key, Math.ceil(windowMs / 1000));
    }

    if (current > maxRequests) {
      const ttl = await redis.ttl(key);
      return res.status(429).json({
        error: 'Too many requests',
        resetTime: ttl
      });
    }

    res.setHeader('X-RateLimit-Limit', maxRequests);
    res.setHeader('X-RateLimit-Remaining', Math.max(0, maxRequests - current));

    next();
  } catch (error) {
    // Si Redis falla, permitir el request pero loguear el error
    console.error('Rate limiter error:', error);
    next();
  }
};

module.exports = rateLimiter;
```

5.6 Logger Utility

```
// services/telegram-bot/src/utils/logger.js
const winston = require('winston');
const path = require('path');

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
```

```
    winston.format.timestamp(),
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  defaultMeta: { service: 'telegram-bot' },
  transports: [
    new winston.transports.File({
      filename: path.join(__dirname, '../../logs/error.log'),
      level: 'error'
    }),
    new winston.transports.File({
      filename: path.join(__dirname, '../../logs/combined.log')
    })
  ]
});

if (process.env.NODE_ENV !== 'production') {
  logger.add(new winston.transports.Console({
    format: winston.format.combine(
      winston.format.colorize(),
      winston.format.simple()
    )
  }));
}

module.exports = logger;
```

6. Configuración y Ejecución

6.1 Script de Setup

```
#!/bin/bash
# scripts/setup.sh

set -e

echo "🌤️ Configurando asistente meteorológico..."

# Verificar dependencias
command -v docker >/dev/null 2>&1 || { echo "❌ Docker no está instalado"; exit 1; }
command -v node >/dev/null 2>&1 || { echo "❌ Node.js no está instalado"; exit 1; }

# Crear .env si no existe
if [ ! -f .env ]; then
    echo "📝 Creando archivo .env..."
    cp .env.example .env
    echo "⚠️  IMPORTANTE: Edita el archivo .env con tus API keys"
    code .env 2>/dev/null || nano .env
fi

# Crear directorio de logs
echo "📁 Creando directorios de logs..."
mkdir -p services/telegram-bot/logs
```

```
mkdir -p services/route-processor/logs
mkdir -p services/weather-orchestrator/logs
mkdir -p services/response-composer/logs

# Instalar dependencias
echo "📦 Instalando dependencias..."
npm install

# Instalar dependencias de servicios
for service in services/*/; do
    if [ -f "$service/package.json" ]; then
        echo "📦 Instalando dependencias para $(basename "$service")..."
        (cd "$service" && npm install)
    fi
done

# Construir imágenes Docker
echo "🐳 Construyendo imágenes Docker..."
docker-compose build

echo "✅ Setup completado!"
echo ""
echo "Próximos pasos:"
echo "1. Editar .env con tus API keys"
echo "2. Ejecutar: npm run dev"
echo "3. Probar el bot en Telegram"
```

6.2 Script de Deploy

```
#!/bin/bash
# scripts/deploy.sh

set -e

echo "🚀 Desplegando asistente meteorológico..."

# Verificar variables de entorno
required_vars=("BOT_TOKEN" "METEOBLUE_API_KEY" "AEMET_API_KEY" "ORS_API_KEY")

for var in "${required_vars[@]}"; do
    if [[ -z "${!var}" ]]; then
        echo "❌ Error: Variable $var no está configurada en .env"
        exit 1
    fi
done

# Parar servicios existentes
echo "🛑 Parando servicios existentes..."
docker-compose down

# Construir nuevas imágenes
echo "🔨 Construyendo imágenes..."
docker-compose build --no-cache

# Iniciar servicios
```

```
echo "🚀 Iniciando servicios..."
docker-compose up -d

# Esperar que los servicios estén listos
echo "⏳ Esperando que los servicios estén listos..."
sleep 15

# Verificar salud de servicios
echo "🔍 Verificando salud de servicios..."
for port in 3000 3001 3002 3003; do
    if curl -f http://localhost:$port/health > /dev/null 2>&1; then
        echo "✅ Servicio en puerto $port: OK"
    else
        echo "❌ Servicio en puerto $port: FALLO"
    fi
done

# Configurar webhook de Telegram si está configurado
if [[ -n "$WEBHOOK_URL" && "$NODE_ENV" == "production" ]]; then
    echo "🔗 Configurando webhook de Telegram..."
    curl -X POST "https://api.telegram.org/bot${BOT_TOKEN}/setWebhook" \
         -d "url=${WEBHOOK_URL}/webhook" \
         -d "max_connections=40"
    echo ""
fi

echo "✅ Despliegue completado!"
echo ""
echo "📊 Ver logs: docker-compose logs -f"
echo "📊 Ver estado: docker-compose ps"
echo "🤖 Probar bot: busca tu bot en Telegram"
```

6.3 Comandos de Ejecución

```
# 1. Setup inicial
chmod +x scripts/*.sh
./scripts/setup.sh

# 2. Configurar variables de entorno
cp .env.example .env
# Editar .env con tus API keys

# 3. Ejecutar en desarrollo
npm run dev

# 4. Ver logs en tiempo real
docker-compose logs -f telegram-bot

# 5. Parar servicios
docker-compose down

# 6. Ejecutar tests
npm test
```

```
# 7. Deploy a producción
./scripts/deploy.sh
```

7. Testing y Validación

7.1 Verificar Configuración

```
# Verificar que Redis está funcionando
docker exec -it weather-redis redis-cli ping

# Verificar APIs de servicios
curl http://localhost:3000/health
curl http://localhost:3001/health
curl http://localhost:3002/health
curl http://localhost:3003/health

# Ver logs de un servicio
docker-compose logs telegram-bot
```

7.2 Pruebas del Bot

1. **Buscar tu bot** en Telegram usando el username
2. **Iniciar conversación** con `/start`
3. **Probar mensaje simple**: "Ruta por La Pedriza mañana"
4. **Probar archivo GPX**: Subir un archivo .gpx de prueba

7.3 Pruebas de APIs

```
# Test meteoblue API
curl "https://my.meteoblue.com/packages/basic-1h?lat=40.7&lon=-3.7&apikey=TU_API_KEY"

# Test AEMET API
curl "https://opendata.aemet.es/opendata/api/valores/climatologicos/inventarioestaciones/

# Test OpenRouteService API
curl -X POST "https://api.openrouteservice.org/v2/directions/foot-walking" \
     -H "Authorization: TU_API_KEY" \
     -H "Content-Type: application/json" \
     -d '{"coordinates":[[8.681495,49.41461],[8.687872,49.420318]]}'
```

7.4 Solución de Problemas Comunes

Bot no responde:

- Verificar BOT_TOKEN en .env
- Verificar webhook configurado correctamente

- Comprobar logs: `docker-compose logs telegram-bot`

Error de APIs:

- Verificar API keys válidas
- Comprobar límites de rate limiting
- Verificar conectividad de red

Servicios no inician:

- Verificar puertos libres
- Comprobar configuración Docker
- Revisar variables de entorno

¡Proyecto Listo!

Una vez completados todos los pasos, tendrás un asistente meteorológico completamente funcional que puede:

- ✅ Procesar descripciones de rutas en texto natural
- ✅ Analizar archivos GPX
- ✅ Consultar múltiples fuentes meteorológicas
- ✅ Generar briefings detallados con avisos de riesgo
- ✅ Proporcionar enlaces directos a más información

¡A disfrutar de tus rutas con total seguridad meteorológica!