Zona horaria: el servidor va en UTC y la aplicación declara la suya

Nuestros servidores funcionan en **UTC**, y eso es una decisión deliberada, no un descuido de configuración. Cada aplicación alojada declara la zona horaria que necesita.

Es una fuente habitual de confusión, sobre todo en aplicaciones que programan envíos o tareas para una hora concreta: correos que salen dos horas tarde, informes que se generan a destiempo, avisos que llegan cuando ya no tocaba. Esta página explica por qué el servidor va en UTC, y sobre todo **dónde hay que declarar la zona horaria para que el cambio surta efecto**, que casi nunca es donde uno mira primero.

Por qué el servidor va en UTC

UTC no cambia nunca. No tiene horario de verano, no adelanta ni atrasa, y no depende del país.

Un servidor compartido aloja aplicaciones de clientes que pueden necesitar zonas distintas. Si el servidor se fijara en `Europe/Madrid`, todas heredarían esa zona y las que necesitasen otra tendrían que pelearse contra ella. Además, dos veces al año el reloj daría un salto: en octubre una hora se repite y en marzo una hora no existe. Toda tarea programada en esa franja se ejecuta dos veces o ninguna.

Con el servidor en UTC ese problema desaparece de la base, y cada aplicación resuelve su horario local por su cuenta.

Hay dos relojes, no uno

Esta es la parte que se pasa por alto y la que explica la mayoría de los casos.

Una aplicación PHP con base de datos tiene **dos relojes independientes**:

- El de **PHP**, que es el que devuelve `date()` y compañía.
- El de **la base de datos**, que es el que devuelve `NOW()` en una consulta SQL.

Cambiar uno **no** cambia el otro. Y muchas aplicaciones no deciden con PHP: deciden con SQL. Cuando una aplicación selecciona qué enviar con una consulta del estilo

```
SELECT ... WHERE fecha_programada <= NOW()
```

quien decide la hora es la base de datos. Se puede poner PHP en la zona que se quiera, que el envío seguirá saliendo a la hora antigua.

Regla práctica: si lo que falla es *cuándo ocurre algo programado*, sospecha del reloj de la base de datos antes que del de PHP.

Cómo comprobar cada reloj

El de PHP, desde tu terminal SSH:

```
php -r 'echo date("Y-m-d H:i:s T P"), PHP_EOL;'
```

Si usas una versión concreta en el cron, comprueba **esa misma**, invocándola por su ruta completa igual que hace el cron. Distintas versiones de PHP pueden tener configuraciones distintas.

El de la base de datos, desde el cliente MySQL o desde phpMyAdmin:

```
SELECT @@session.time_zone, NOW(), UTC_TIMESTAMP();
```

Si `NOW()` y `UTC_TIMESTAMP()` devuelven lo mismo, tu sesión está en UTC.

Declarar la zona en PHP

Lo correcto es hacerlo **dentro de tu aplicación**, no en la configuración del servidor.

La forma más directa, al principio del arranque de la aplicación:

```
date_default_timezone_set('Europe/Madrid');
```

Muchas aplicaciones traen ya su propio ajuste de zona horaria en su panel o en su fichero de configuración. **Si existe, úsalo**: suele hacer más cosas que llamar a esa función, como veremos enseguida.

También puedes fijarla para todo un dominio con un fichero `.user.ini` en la raíz del sitio:

```
date.timezone = "Europe/Madrid"
```

Ten en cuenta que ese fichero **no afecta a las ejecuciones por línea de comandos**, así que no sirve para tus tareas del cron. Para el cron, o lo declara la aplicación, o se pasa en la propia orden:

```
/usr/local/php83/bin/php -d date.timezone=Europe/Madrid
/home/<usuario>/domains/<dominio>/public_html/tarea.php
```

Lo que no hay que hacer

No intentes modificar el `php.ini` general del servidor. No vas a poder —tu shell trabaja aislado y esa ruta es de sólo lectura para ti— y, aunque pudieras, sería contraproducente por dos motivos: afectaría a todos los clientes del servidor, y el panel regenera ese fichero en cada actualización de PHP, de modo que el cambio se perdería en la siguiente compilación.

Si te encuentras un mensaje de `Read-only file system` al intentarlo, no es una avería: es el aislamiento previsto de tu cuenta.

Declarar la zona en la base de datos

Se hace por conexión, con una sentencia al abrirla:

```
SET time_zone = 'Europe/Madrid';
```

A partir de ahí, `NOW()` devuelve hora española en esa conexión, y respeta el horario de verano.

Nuestros servidores tienen cargado el catálogo completo de zonas horarias, así que puedes usar nombres como `Europe/Madrid`. Puedes comprobarlo así:

```
SET time_zone = 'Europe/Madrid';
SELECT NOW(), UTC_TIMESTAMP();
```

Si las dos horas se diferencian en una o dos horas según la época del año, funciona correctamente.

Usa siempre el nombre de la zona, nunca un desplazamiento fijo. Un `SET time_zone = '+02:00'` parece equivalente y no lo es: acertará en verano y fallará en invierno, cuando España está en `+01:00`. El nombre conoce las transiciones; el número, no.

El caso de phpList

phpList es el ejemplo perfecto de esto, porque decide con SQL y porque su ajuste tiene un nombre que es fácil equivocar.

En su fichero `config/config.php` la constante correcta es:

```
define('SYSTEM_TIMEZONE', 'Europe/Madrid');
```

Esa constante hace dos cosas a la vez: ajusta el reloj de PHP **y** el de la base de datos. Es exactamente el mecanismo que las alinea, y por eso funciona donde un `date_default_timezone_set()` suelto no llega.

Cuidado con el nombre. `SERVER_TIMEZONE` **no existe** en phpList: si la defines, no da ningún error, simplemente no hace nada, y el problema sigue igual sin ninguna pista de por qué.

Antes de aplicar el cambio: revisa lo que tengas programado

Este aviso vale para phpList y para cualquier aplicación con tareas o envíos programados.

Al corregir la zona horaria, el reloj con el que la aplicación compara sus fechas **se mueve de golpe**. En España, en horario de verano, avanza dos horas. Eso significa que todo lo que tengas pendiente de salir **se disparará dos horas antes** de lo que la pantalla te indica ahora mismo.

Y hay un caso que conviene mirar con especial atención: si mientras el problema estaba activo compensaste el desfase a mano —programando las cosas dos horas más tarde para que salieran a la hora buena—, esas quedarán **mal en sentido contrario** y saldrán con dos horas de retraso.

La recomendación es sencilla: repasa la lista de tareas o campañas pendientes **antes** de tocar la configuración, y reajusta las que hicieran falta justo después.

Resumen

- El servidor va en UTC a propósito, y no se cambia.
- La zona horaria la declara tu aplicación, no el servidor.
- Hay dos relojes independientes: el de PHP y el de la base de datos. Cambiar uno no cambia el otro.
- Si lo que falla es *cuándo ocurre* algo programado, mira el reloj de la base de datos.
- Usa siempre nombres de zona (`Europe/Madrid`), nunca desplazamientos fijos (`+02:00`).
- Si tu aplicación trae su propio ajuste de zona, úsalo: normalmente alinea los dos relojes a la vez.
- Revisa lo que tengas programado antes de aplicar el cambio.

Si después de esto sigues viendo un desfase, abre un ticket indicando qué aplicación es, qué hora esperas y qué hora obtienes, y lo revisamos.

Revision #1
Created 2026-09-05 12:04:53 UTC by Abkrim
Updated 2026-09-05 12:04:53 UTC by Abkrim