GUÍA PARA INGENIEROS DE RED

Servicios y plantillas: ver, probar y extender sin desarrollo

Cómo ver qué genera cada servicio en tus equipos, probar un cambio antes de guardarlo y agregarle lo que la plantilla no trae. Tú trabajas con parámetros; Oktavia se encarga de la plantilla, la compuerta y las pruebas.

Desarrollado por Oktanet | México · 2026 | Documento público

CONTENIDO

1. Para quién es esta guía
2. La idea: parámetros, no Jinja
3. Los servicios de tu dominio
4. La página del servicio
5. Valores declarados y «Ver qué cambia»
6. Probar con un equipo
7. Explícame este servicio
8. IOS XR: sin Jinja, con XML
9. Diccionario NETCONF: buscar como en la CLI
10. Constructor de servicios
11. Extender un servicio
12. La compuerta antes de publicar
13. Las pruebas de red de una extensión
14. Servicios personalizados
15. Con Argos: Servicios por dominio
16. Recetas
17. Cómo funcionan las plantillas y los valores
18. Con Ansible y la API
19. Buenas prácticas

1. Para quién es esta guía

Para ingenieros de red y de automatización, de cliente o de socio, que mantienen los servicios de una red en Oktavia: Campus, WAN, Seguridad y Gestión en una empresa; Transporte, Servicios, Seguridad y Gestión en un proveedor.

Al terminar sabrás leer un servicio de punta a punta, probar un valor contra un equipo real sin guardar nada, y agregar a un servicio lo que su plantilla no trae, sin escribir Jinja y sin pedir desarrollo.

Esta guía complementa la Guía de intención, que explica los alcances y el ciclo de cambio. Si es tu primera vez con Oktavia, empieza por esa.

2. La idea: parámetros, no Jinja

Cada servicio de Oktavia tiene una plantilla base, probada, que convierte valores en configuración. Esa plantilla **no se edita**: se **extiende**. Así una actualización de Oktavia no pisa tu trabajo y tu trabajo no rompe la plantilla.

Tu trabajo como ingeniero está en los **parámetros**: qué valores, en qué alcance y con qué prueba de red. La plantilla de una extensión la trae el catálogo de Oktavia o la propone Argos, y una compuerta determinista decide si se puede publicar.

PIEZA	QUIÉN LA HACE	DÓNDE VIVE
Plantilla base	Oktavia.	El catálogo de servicios. No se toca.
Extensión	El catálogo de Oktavia, con un clic, o Argos a partir de tu descripción.	Junto al servicio que extiende.
Valores	Tú.	La intención (`service_vars`), en el alcance que elijas.
Pruebas	Oktavia las saca de la plantilla y de los parámetros.	El cumplimiento de cada auditoría.

Guardar valores no toca la red. Llegan a los equipos con el ciclo de siempre: auditar, cumplimiento, Gemelo digital, tú apruebas y la verificación lo confirma.

3. Los servicios de tu dominio

Abre **Servicios** y elige el dominio. La pestaña que antes se llamaba Intención ahora se llama **Servicios**: lista todos los servicios del dominio, estén declarados o no.

COLUMNA	QUÉ DICE
Plataforma	Para qué sistemas existe el servicio: `ios` , `iosxe` , `xr` …
Declarado en	Los alcances donde tiene valores.
Equipos	A cuántos equipos les aplica.
Fallas y %	Su cumplimiento en la última auditoría.
Pruebas	Cuántas pruebas de config y de estado lo validan; «sin pruebas» si ninguna.

Los servicios no declarados salen al final, con un enlace para declararlos. El **Resumen** del dominio también muestra la plataforma de cada servicio.

4. La página del servicio

Cada servicio de la lista abre su página. Reúne en un solo lugar todo lo que necesitas para entenderlo y cambiarlo:

1. **Valores declarados**: cada alcance donde se declara, con sus valores, y los botones para editarlos, ver la plantilla y extender (sección 5).

2. **Probar con un equipo**: valores, plantilla y resultado, lado a lado, para un equipo real (sección 6).
3. ✦ **Explícame este servicio**: Argos te explica qué hace y qué genera (sección 7).
4. ＋ **Agregar parámetros** (Cambiar un servicio): lo que puedes agregarle (sección 11).
5. **Equipos donde está activo**, por sitio, con ⚠ en los que fallan, y **Pruebas que lo validan**.

Las pruebas son de dos tipos. Las de **config** comparan la running con lo que genera la plantilla, una regla por equipo. Las de **estado** comprueban que el servicio funcione en la red.

5. Valores declarados y «Ver qué cambia»

El plegable **Valores declarados** muestra cada alcance donde vive el servicio — `defaults`, `by_role`, `by_site`, `by_device`, `by_model`, `by_platform` …— con sus valores. En la misma fila están **Editar valores**, **Plantilla** y **Extender aquí**.

Editar y ver qué cambia

1. Pulsa **Editar valores** en el alcance que quieres cambiar. Se abre un editor JSON.
2. Cambia los valores.
3. Pulsa **Ver qué cambia**. Oktavia muestra, por equipo, las líneas de la configuración deseada que cambiarían: `−` lo de hoy, `+` lo que queda con tu cambio. Los secretos salen ocultos.
4. Si es lo que esperabas, pulsa **Guardar**. Si no, ajusta y vuelve a ver.

```
r1
  router bgp 65001
+  neighbor 198.51.100.1 timers 10 30
```

Son líneas reales de configuración, no sólo valores. Si un equipo que no esperabas aparece en la lista, tu alcance es más amplio de lo que creías: no guardes todavía.

> **Qué necesitas saber.** Editar requiere ser administrador de la organización. Los secretos van siempre como referencia a la bóveda, `<<vault:…>>`; nunca en claro.

6. Probar con un equipo

Es la mejor forma de entender un servicio. Elige un equipo en el menú, agrupado por sitio, y verás tres columnas lado a lado:

COLUMNA	QUÉ MUESTRA
1. Valores	Los valores del equipo para ese servicio y de qué alcances salen. Cada alcance es un botón que abre su declaración para editarla.
2. Plantilla	La plantilla Jinja del servicio.
3. Lo que genera	La configuración que genera hoy en ese equipo.

La plantilla, el JSON y el XML llevan colores como en VS Code. En la columna 3, lo que sale de los parámetros del equipo va **resaltado**; al pasar el mouse te dice de qué parámetro sale. Lo que no está resaltado es fijo de la plantilla.

Probar estos valores

En la columna 1 puedes cambiar los valores y pulsar **Probar estos valores**. Oktavia renderiza con el mismo motor de las auditorías y te muestra la diferencia. **No guarda nada.** Úsalo para responder «¿qué pasaría si…?» antes de tocar la intención.

> **Si la prueba dice «no cambia nada».** Algunos valores salen del inventario del equipo, por ejemplo `ospf_config.router_id`, y ganan sobre los del servicio. Si cambias uno de esos en el servicio, la configuración no se mueve. Revisa el inventario del equipo.

7. Explícame este servicio

Pulsa ✦ **Explícame este servicio** y Argos, con IA local o con Claude, te explica:

- qué hace el servicio;
- cada parámetro;
- qué genera en el equipo que elegiste;
- consejos para usarlo bien.

Argos recibe la plantilla, los valores enmascarados y lo generado. Nunca recibe secretos ni la running completa del equipo.

8. IOS XR: sin Jinja, con XML

En IOS XR no hay plantilla Jinja. El carril NETCONF arma el XML YANG desde los mismos valores. La página del servicio funciona igual, con estas diferencias:

- **Columna 2**: explica cómo el carril NETCONF construye la configuración.
- **Columna 3**: muestra el XML del equipo elegido. Los valores del equipo también van resaltados, así distingues lo que sale de variables de lo fijo del carril.
- **Ver qué cambia** y **Probar estos valores** comparan el plan NETCONF de hoy contra el de tu cambio, tal como lo empujaría el Gemelo digital, con *commit confirmed*.

Aplica a los servicios del núcleo y a los de cliente. Hay una receta en la sección 16.

9. Diccionario NETCONF: buscar como en la CLI

Para extender un servicio de IOS XR no necesitas saber NETCONF. **Servicios → Diccionario NETCONF** tiene los parámetros que acepta el equipo, leídos de sus propios modelos YANG, con su tipo, rango y valor por omisión.

1. **Busca como en la CLI**: «neighbor timers keepalive», «isis lsp-gen-interval», «interface mtu». También puedes pegar una línea de XR, como `neighbor 192.0.2.1 timers 10 30`: Oktavia ignora los valores y encuentra las hojas.
2. **Elige el parámetro.** Cada resultado se muestra como en la CLI —«router bgp ‹as› › neighbor ‹ip› › timers › keepalive-interval»— y las variantes (VRF, session-group…) quedan agrupadas.
3. **Usar en un servicio.** Oktavia arma la extensión sin IA: el XML con su contexto (el AS sale de BGP del núcleo y el proceso de IS-IS), el campo con su rango y la baja. Tú sólo pones el valor, con la vista previa y «Guardar y proponer».

Si ya tienes el XML que quieres en el equipo, **Desde XML** lo convierte en parámetros: cada hoja con texto es un campo, los hermanos repetidos son una lista y lo que parece secreto queda como `<<vault:…>>`.

10. Constructor de servicios: arma el servicio con las opciones del equipo

Servicios → Constructor de servicios reúne en una sola pantalla lo que antes eran varios pasos: eliges las opciones que el equipo acepta, Oktavia arma la plantilla y el formulario, ves el resultado en CLI de IOS XR y en XML NETCONF, y lo publicas con su vista previa. No escribes modelos YANG ni paquetes de servicio: las opciones salen del YANG del propio equipo (el mismo índice del Diccionario NETCONF).

Camino IOS XR · NETCONF, paso a paso

Ejemplo: keepalive y hold de BGP por vecino en los PE.

1. **Busca como en la CLI**: «neighbor timers keepalive», o pega una línea de XR como `neighbor 192.0.2.1 timers 10 30`. Marca *keepalive-interval* y *holdtime*; aparecen como «router bgp ‹as› › neighbor ‹ip› › timers».
2. **Elige el servicio base y el alcance**: «BGP del núcleo (IOS XR)» en el rol `pe`. El AS sale del servicio base; no lo vuelves a escribir.
3. **Llena el formulario**. Cada opción es un campo con su rango (0..65535) y la lista de vecinos llega prellenada con los vecinos reales del alcance; sólo pones 10 y 30.
4. **Mira el resultado** en las pestañas de la derecha: *CLI de XR*, *XML NETCONF* y *Traducir*.
5. **Guardar y proponer el cambio.** Pasa por la compuerta (sintaxis, secretos, impacto en otros servicios), muestra por equipo lo que cambia y propone el cambio en el Gemelo: el dry-run hace

validate en los equipos y tú apruebas y confirmas con commit confirmado.

```
router bgp 64500
neighbor 192.0.2.1
 timers 10 30
neighbor 198.51.100.1
 timers 10 30
```

```
<router xmlns="http://cisco.com/ns/yang/Cisco-IOS-XR-um-router-bgp-cfg">
  <bgp><as><as-number>64500</as-number>
    <neighbors>
      <neighbor><neighbor-address>192.0.2.1</neighbor-address>
        <timers><keepalive-interval>10</keepalive-interval><holdtime>30</holdtime></timers>
      </neighbor>
    </neighbors>
  </as></bgp>
</router>
```

Traducir CLI ⇄ XML

La pestaña *Traducir* convierte CLI de IOS XR en el XML NETCONF que la representa, y al revés. La traducción es **determinista**: sale del YANG del equipo (los modelos `Cisco-IOS-XR-um-*`), no de una IA. Con las configuraciones reales del laboratorio, XML → CLI → XML regresa idéntico en IS-IS, BGP, L2VPN, QoS, Segment Routing y telemetría. Lo que el diccionario no conoce se avisa, no se inventa. *Próximamente:* la traducción «en vivo», en la que el propio equipo convierte la CLI sin hacer commit y descarta el resultado.

Camino IOS / IOS-XE · CLI

1. **Describe lo que quieres** («IP SLA hacia el proveedor en los routers WAN») y, si tienes, pega la config de un equipo que ya lo tenga. Las líneas con secretos se quitan antes de mandarlas a la IA.
2. **Argos redacta la plantilla Jinja** (IA local o Claude) con sus parámetros.
3. Mismo formulario, misma vista previa por equipo y el mismo botón **Guardar y proponer**.

El editor anterior sigue disponible como *Editor avanzado* para quien quiera escribir la plantilla a mano.

11. Extender un servicio

Cuando la plantilla base no trae lo que necesitas, la extiendes. Tienes dos entradas:

- **Opciones avanzadas**: extensiones del catálogo de Oktavia, probadas, que se activan con un clic. Por ejemplo, en OSPF: timers, BFD y tipos de área; en BGP: timers por vecino, address-family y agregados; en interfaces: `bandwidth`.
- **＋ Agregar parámetros**, en la fila del alcance donde quieres los valores, o desde «Probar con un equipo».

Cambiar un servicio: una pregunta, un formulario y un botón

1. **¿Qué quieres cambiar?** Escríbelo con tus palabras, por ejemplo «keepalive 10 y hold 30 a los vecinos del RR». Si coincide con una extensión del catálogo de Oktavia se elige sola, sin IA; si no, Argos la genera (IA local o Claude). Los atajos del catálogo siguen abajo como botones.
2. **Formulario.** Los campos salen de la extensión, con etiqueta, rango y valor por omisión; las listas son tablas con «+ fila», y viene prellenado con lo real del alcance, como tus vecinos. El JSON y la plantilla quedan en «Avanzado».
3. **Vista previa automática.** Al cambiar un valor válido ves, por equipo, las líneas que cambian (en IOS XR, el XML que agrega). Si la compuerta rechaza algo, lo dice en una línea.
4. **Guardar y proponer el cambio.** Un solo botón: publica la extensión, deja los valores en el alcance y propone el cambio en el Gemelo Digital. Se habilita sólo cuando la última vista previa pasó y corresponde a los valores de ahora.

La misma experiencia está en Argos: **Orquestación personalizada → Cambiar un servicio**, o escribe «cambia un servicio».

Los valores de la extensión quedan en la intención (`service_vars`), en el mismo alcance donde pulsaste **+ Agregar parámetros**. Llegan a los equipos con el ciclo: auditar → cumplimiento → Gemelo digital → tú apruebas → verificación.

12. La compuerta antes de publicar

Antes de publicar una extensión, una compuerta determinista la revisa. No es una opinión de la IA: son reglas fijas.

REGLA	QUÉ REVISA
Comandos bloqueados	La plantilla no lleva comandos prohibidos.
Ningún secreto literal	Tampoco tipo 7. Los secretos van como `<<vault:…>>` .
Baja limpia	Quitar la extensión sólo usa `no` o `default` .
Colateral	Si la extensión cambia líneas de otro servicio, no se publica.

La vista previa renderiza la extensión en equipos reales sin tocar la red, y queda firmada: **sólo se publica lo que pasó la compuerta con esos mismos valores**.

13. Las pruebas de red de una extensión

Una extensión no termina en la configuración: también dice cómo comprobar que funciona. Hay dos tipos de prueba.

Pruebas de estado

Comprueban que algo esté arriba: vecinos BGP u OSPF, interfaces, BFD, IS-IS, rutas. Se sacan de los parámetros. Por ejemplo, de `neighbors[].address` sale la lista de vecinos que deben estar establecidos.

Pruebas de valor

Comparan lo que se ve en la red contra lo que declaraste. No basta con que el vecino esté arriba: también tiene que tener el valor correcto.

- Timers BGP negociados con cada vecino.
- Hello y dead de OSPF por interfaz.
- Usuarios SNMPv3: grupo y protocolos, sin claves.

Las extensiones del catálogo `bgp_timers` y `ospf_timers` ya traen su prueba de valor. Al auditar aparece, por ejemplo, «BGP · timers negociados con el vecino». Hoy las pruebas de valor existen en IOS e IOS XE.

14. Servicios personalizados

Cuando lo que necesitas no extiende un servicio existente, crea uno nuevo en **Servicios → Servicios personalizados**.

1. Describe el servicio o pega una configuración de ejemplo.
2. Argos propone la plantilla, los campos y un ejemplo de valores.
3. La misma compuerta de la sección 12 decide si se puede publicar.

Desde ahí también puedes abrir **Ver plantilla** de cualquier servicio, para leerla sin editarla.

15. Con Argos: Servicios por dominio

Si prefieres pedirlo con tus palabras, abre **Argos → Configurar → Servicios por dominio**. Lo que antes era el asistente del Configurador ahora vive aquí.

1. Elige el dominio y el servicio. Sólo aparecen los de las plataformas de tu red.
2. Di qué valores quieres y dónde, por ejemplo `timers 10 30 en los wan_router`. También puedes traerlos del respaldo de un equipo.
3. Argos propone el cambio. Pulsa 🔍 **Qué cambia en los equipos** para ver la configuración deseada por equipo antes de guardar.
4. Si es lo que querías, pulsa **Aplicar y cerrar el ciclo**. Tú sigues aprobando el cambio en el Gemelo digital.

16. Recetas

BGP · Timers 10/30 en un vecino

Quieres detectar antes la caída de un proveedor. La plantilla de BGP no trae timers por vecino; la extensión `bgp_timers` del catálogo sí.

1. En **Servicios** → **WAN**, abre el servicio BGP.
2. En **Valores declarados**, busca el alcance del router, normalmente `by_device`, y pulsa **+ Agregar parámetros**.
3. Elige **BGP · timers por vecino** del catálogo y llena los parámetros:

```
{
  "neighbors": [
    { "address": "198.51.100.1", "keepalive": 10, "hold": 30 }
  ]
}
```

4. Pulsa **Ver qué cambia**. Debe aparecer una sola línea nueva, en ese equipo:

```
r1
  router bgp 65001
+  neighbor 198.51.100.1 timers 10 30
```

5. Guarda y cierra el ciclo. Aprueba el cambio en el Gemelo digital.
6. Después de la verificación, revisa el cumplimiento del equipo: aparece la prueba de valor **BGP · timers negociados con el vecino**. El vecino negocia el hold menor: si el otro lado tiene uno más bajo, eso es lo que verás.

Interfaces · Agregar `bandwidth`

La plantilla de interfaces no trae `bandwidth`, y lo necesitas para el costo de OSPF o EIGRP y para un QoS en porcentaje. `bandwidth` no cambia la velocidad real del enlace.

1. Abre el servicio de interfaces del router y, en **Opciones avanzadas**, activa **Interfaces · bandwidth** (`interface_bandwidth`).
2. Llena una entrada por interfaz, en kbps:

```
{
  "interfaces": [
    { "name": "GigabitEthernet4", "bandwidth_kbps": 50000 }
```

```
  ]
}
```

3. En **Probar con un equipo**, elige el router y revisa la columna 3. Lo resaltado sale de tus parámetros:

```
interface GigabitEthernet4
 bandwidth 50000
!
```

4. Guarda y cierra el ciclo. La prueba de estado de la extensión comprueba que la interfaz siga arriba.

Un servicio nuevo con Argos desde una config de ejemplo

Quieres una sonda IP SLA hacia el gateway del proveedor en cada router de sucursal, y no hay un servicio para eso.

1. Abre **Servicios → Servicios personalizados** y pega la configuración que ya usas en un equipo:

```
ip sla 10
 icmp-echo 198.51.100.1 source-interface GigabitEthernet2
 frequency 30
ip sla schedule 10 life forever start-time now
```

2. Argos propone la plantilla, los campos (por ejemplo, número de sonda, destino, interfaz de origen y frecuencia) y un ejemplo de valores. Revisa que lo variable sea un campo y lo fijo quede fijo.

3. Revisa la vista previa con dos o tres routers. Si la compuerta rechaza algo, corrígelo antes de seguir.

4. Publica, declara los valores en el alcance que corresponda, por ejemplo `by_role`, y cierra el ciclo.

IOS XR · Un cliente nuevo del reflector BGP

Vas a sumar un PE nuevo como cliente del route reflector y quieres ver el XML que cambia antes de guardar.

1. En **Servicios → Transporte**, abre el servicio de BGP del núcleo (`xr_bgp`).

2. En **Probar con un equipo**, elige el route reflector. La columna 3 muestra su XML, con los valores resaltados.

3. En la columna 1, agrega la loopback del PE nuevo a `clients`. Recuerda: una lista se reescribe completa.

```
{
  "asn": 65000,
  "route_reflector": true,
  "clients": ["192.0.2.11", "192.0.2.12", "192.0.2.13"]
}
```

4 Pulsa **Probar estos valores**. Oktavia compara el plan NETCONF de hoy con el de tu cambio y muestra el vecino nuevo:

```
+ <neighbor>
+   <neighbor-address>192.0.2.13</neighbor-address>
+   <remote-as>65000</remote-as>
+   <update-source>Loopback0</update-source>
+   <address-families>
+     <address-family>
+       <af-name>vpnv4-unicast</af-name>
+       <route-reflector-client/>
+     </address-family>
+   </address-families>
+ </neighbor>
```

5 Si es lo esperado, edita la declaración con **Editar valores**, revisa **Ver qué cambia** y guarda.

6 El núcleo cambia como una sola transacción con *commit confirmed*: si no confirmas, los equipos regresan solos.

17. Cómo funcionan las plantillas y los valores

Dos piezas

Cada servicio tiene una **plantilla** por plataforma —Jinja para CLI, XML YANG para NETCONF— y sus **valores** en la intención (`service_vars`). El ingeniero sólo toca los valores; la plantilla la mantiene Oktavia y no se edita: se extiende.

Alcances y quién gana

Los valores se declaran por alcance y se mezclan del más general al más específico: `defaults`, `by_site`, `by_site_type`, `by_role`, `by_platform`, `by_model`, `by_tag`, `by_filter` y `by_device`. Gana el último; los objetos se mezclan campo por campo y las listas se reemplazan completas. La tabla completa está en la Guía de intención.

`by_filter` combina condiciones, y también lee los **atributos clave = valor** del equipo y de su sitio: `attr.tier=oro__role=pe` aplica a los PE con ese atributo y `site_attr.region=norte` a los equipos de los sitios de esa región. Si dos filtros aplican, gana el que tiene más condiciones.

Extensiones

Una extensión declara `extends` con el servicio base, sus campos, su plantilla y su plantilla de baja. Genera su propio bloque: no reescribe la plantilla base ni las líneas de otro servicio (la compuerta lo impide). Su prueba de estado o de valor llega con ella al cumplimiento.

Bajas

Quitar algo de la intención no lo borra solo del equipo. Por eso cada extensión trae su plantilla de baja (`no …` en CLI) y, en IOS XR, la fila dada de baja queda marcada como `retired` y el plan la quita con `nc:operation="remove"`; el cumplimiento espera que ya no esté.

Del valor al equipo

Guardar no toca la red. El cambio llega con el ciclo: el Gemelo Digital propone el diff, una persona aprueba, se aplica (en IOS XR con validación previa y commit confirmado) y la auditoría verifica.

18. Con Ansible y la API

Lo mismo se automatiza desde fuera: un playbook puede cambiar un parámetro o dar de alta un servicio de cliente con la vista previa, guardar la intención y devolver el cambio propuesto en el Gemelo; aprobar y aplicar siguen siendo de una persona. Paso a paso en Automatiza Oktavia con su API, Ansible y Python.

19. Buenas prácticas

- **Prueba antes de guardar.** «Probar estos valores» no guarda nada: úsalo sin miedo.
- **Lee «Ver qué cambia» equipo por equipo.** Si aparece un equipo que no esperabas, revisa el alcance.
- **Prefiere el catálogo.** Una extensión del catálogo ya está probada y trae su prueba de red.
- **Extiende, no copies.** La plantilla base no se edita; tu trabajo vive en la extensión y en los valores.
- **Declara en el alcance más general posible.** Lo mismo que en la intención: `by_device` sólo para lo que es del equipo.
- **Dale a cada extensión su prueba.** Un servicio «sin pruebas» no te avisa cuando deja de funcionar.
- **Nada de secretos en claro.** Todo por la bóveda, con `<<vault:…>>`.
- **Verifica con cumplimiento.** Un cambio termina cuando la auditoría lo confirma.
- **Usa atributos para los grupos que no son sitio ni rol** (nivel de cliente, región, contrato) y decláralos con `by_filter`, en vez de repetir valores por equipo.
- **Nombra bien las extensiones.** El id dice el servicio y lo que agrega (`xr_bgp_timers`, `ospf_bfd`); la etiqueta, en lenguaje de red.
- **Sube la versión al cambiar una extensión publicada.** Oktavia guarda la versión y su origen del catálogo; el cumplimiento y el Gemelo usan siempre la publicada.
- **En el núcleo, primero la validación previa.** En IOS XR prueba el cambio (dry-run) y aplica con commit confirmado: si la verificación no pasa, el equipo revierte solo.

Para los alcances y el ciclo de cambio, sigue con la Guía de intención.

Acerca de Oktanet

Oktanet es una empresa mexicana de automatización de redes con enfoque en operaciones controladas, trazables y escalables para entornos empresariales en Latinoamérica.

Para una demostración sobre su entorno, escríbanos desde oktanet.io.

LinkedIn · **YouTube**

© 2026 Oktanet · Oktavia · Documento público, sin información confidencial de clientes.