Saturn Studio — Mejores Prácticas

Guía completa para desarrollar robots robustos, seguros y mantenibles en Saturn Studio

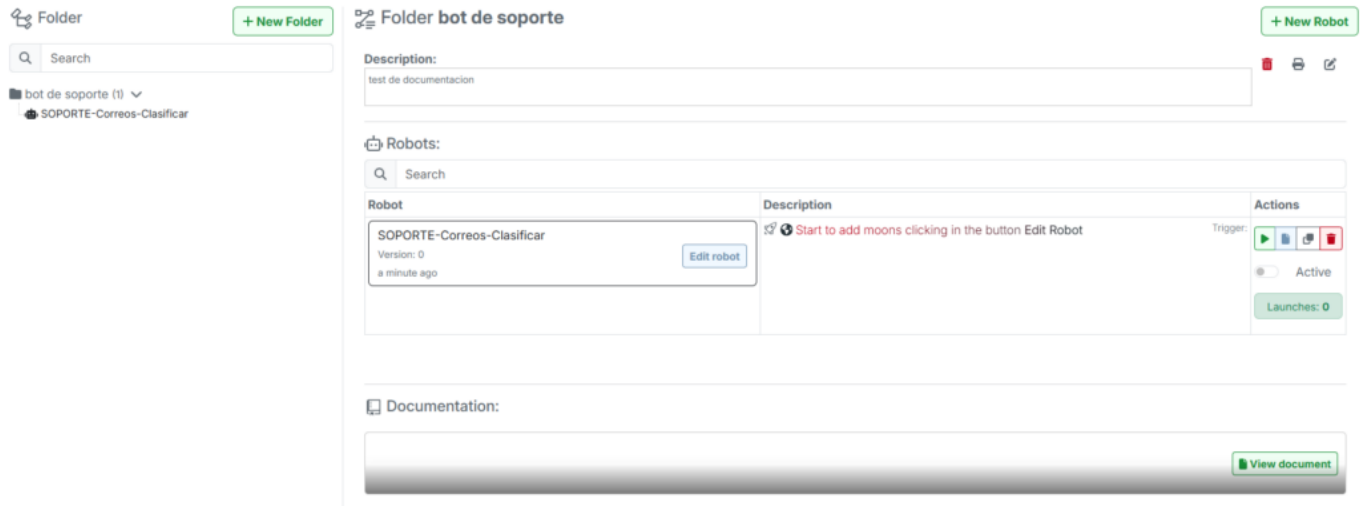
> **Nota: Estructura** — Este documento cubre 8 categorías de buenas prácticas: nomenclatura y organización, variables y datos, manejo de errores, seguridad y credenciales, rendimiento, prompts de IA, colaboración en equipo y paso a producción. Cada práctica incluye el porqué, el cómo y un ejemplo concreto.

1. Nomenclatura y Organización

Un robot bien nombrado y organizado se mantiene, se depura y se comparte mucho más fácilmente. Estas prácticas son el fundamento de cualquier proyecto de automatización profesional.

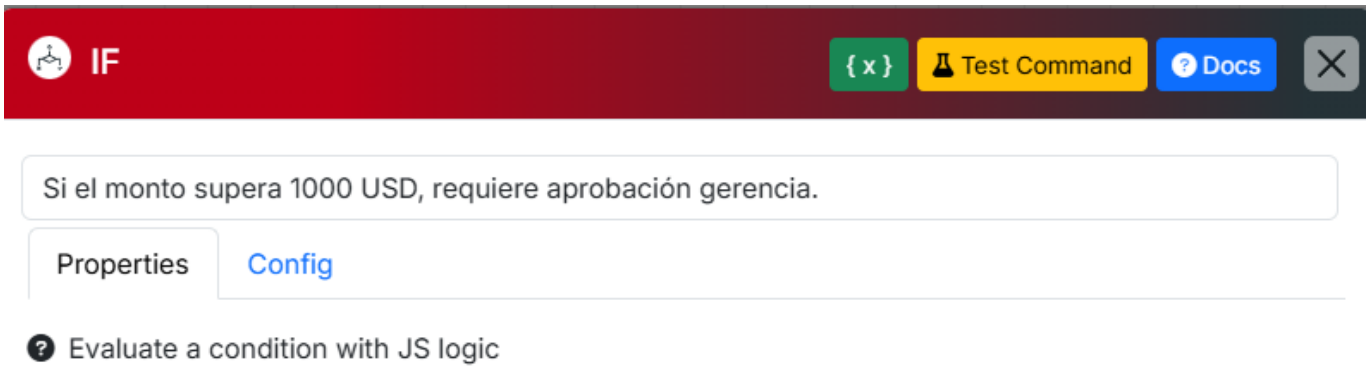
BP-01 | Nombra los robots con un patrón consistente

- **Por qué:** Un nombre claro reduce el tiempo de búsqueda y evita confusión entre robots similares. En equipos grandes, la falta de convención genera duplicados y ambigüedades.
- **Cómo:** Usa el patrón: [AREA]-[PROCESO]-[ACCION]. Evitar nombres genéricos como ‘bot1’ o ‘prueba_final_v3’.
- **Ejemplo:** Un robot que clasifica correos de soporte debería llamarse: SOPORTE-Correos-Clasificar. Así cualquier miembro del equipo entiende su función sin abrirlo.



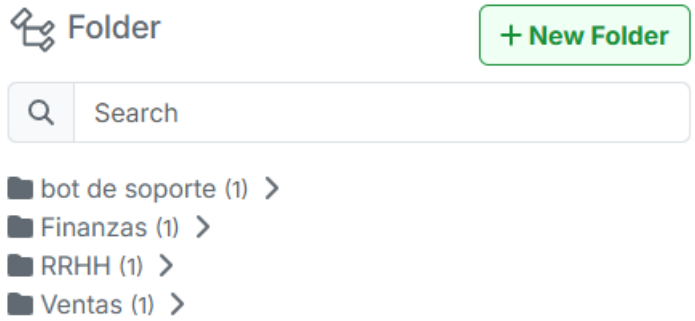
BP-02 | Documenta cada Moon con una descripción breve

- **Por qué:** En un flujo complejo, volver a entender la lógica de una Moon sin descripción puede tomar horas. La documentación inline es la diferencia entre un robot mantenible y uno descartable.
- **Cómo:** Usa el campo de descripción o notas de cada Moon. Una línea es suficiente: # Clasifica el correo en: consulta / reclamo / spam. Para Moons críticas agrega el contexto de por qué se toma esa decisión.
- **Ejemplo:** Una Moon de tipo If/Else sin descripción es un enigma. Con la descripción #Si el monto supera 1000 USD, requiere aprobación gerencia es autoexplicativa.
- ***Si es la primera vez que utiliza las Moons de Saturn Studio***, acceda a la **[**[Guía de introducción y uso de Moons]**]** para aprender a configurarlas.



BP-03 | Organiza los robots en carpetas por área de negocio

- **Por qué:** A medida que crecen los robots, un workspace sin organización se vuelve imposible de navegar. Las carpetas permiten permisos granulares y búsqueda eficiente.
- **Cómo:** Crear carpetas por área: Finanzas, RRHH, Ventas, TI, Operaciones.
- **Ejemplo:** No mezclar robots de Finanzas con robots de RRHH en la misma carpeta. Un nuevo integrante del equipo debe poder ubicar cualquier robot en menos de 30 segundos.



2. Variables y Datos

El manejo correcto de variables es la diferencia entre un robot que funciona en cualquier entorno y uno que solo funciona bajo condiciones específicas.

Nomenclatura de variables

✅ Hacer	❌ No hacer
Usar nombres descriptivos en minúsculas con guion bajo: monto_factura, correo_cliente, resultado_clasificacion	Usar nombres de una letra (a, x, tmp) o abreviaciones ambiguas (mc, rc, res)
Usar MAYÚSCULAS para variables globales y constantes: API_URL, DB_HOST, SLACK_CHANNEL	Mezclar globales y locales con el mismo patrón de nombre
Inicializar las variables críticas al inicio del flujo con valores por defecto	Asumir que las variables estarán vacías o tendrán el valor esperado sin verificarlo
Limpiar variables que contienen datos sensibles (contraseñas, tokens) después de usarlas	Dejar tokens o contraseñas en variables locales durante toda la ejecución del robot

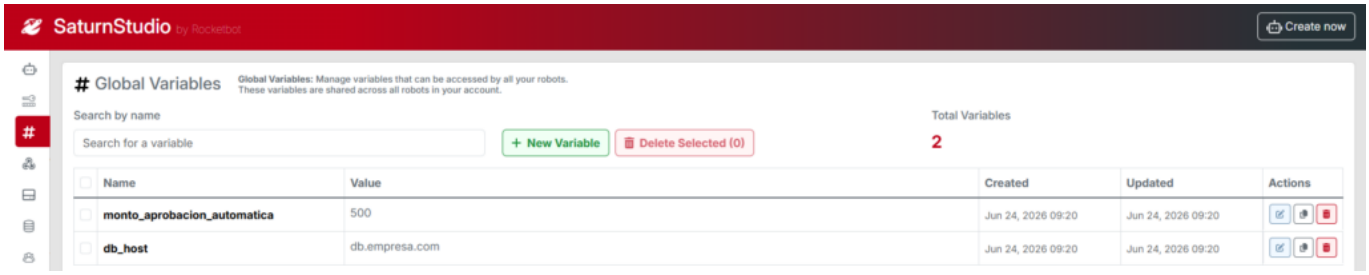
BP-04 | Valida siempre el resultado de una Moon antes de usarlo

- **Por qué:** Un comando puede retornar null, un objeto vacío o un error silencioso. Si el siguiente paso asume que el dato está bien y no lo está, el robot falla de forma inesperada y difícil de depurar.
- **Cómo:** Después de cada Moon que retorna un dato crítico, agregar una Moon If/Else que verifica: ¿el resultado está vacío? ¿es null? ¿tiene el campo esperado? Si falla la validación, terminar con un mensaje de error descriptivo.
- **Ejemplo:** Después de leer correos con Gmail, verificar que el array de resultados no esté vacío antes de intentar acceder a resultado[0]. Si está vacío, loggear ‘Sin correos nuevos’ y terminar el flujo.

BP-05 | Usa Variables Globales para configuración compartida

- **Por qué:** Si diez robots distintos usan la misma URL de API o el mismo host de base de datos, y ese valor cambia, tendrás que editar diez robots. Con una Variable Global, el cambio es en un solo lugar.
- **Cómo:** Crear Variables Globales para: URLs de APIs internas, hostnames de

bases de datos, umbrales de decisión (ej: monto_aprobacion_automatica = 500), emails de notificación de errores.
- **Ejemplo:** Variable Global db_host = 'db.empresa.com'. Si el servidor migra, solo se actualiza esa variable y los 15 robots que la usan se adaptan automáticamente sin editar ningún flujo.
- ***Si es la primera vez que utiliza variables globales***, acceda a la **[Guía de introducción y uso de Variables Globales]** para aprender a configurarlas.

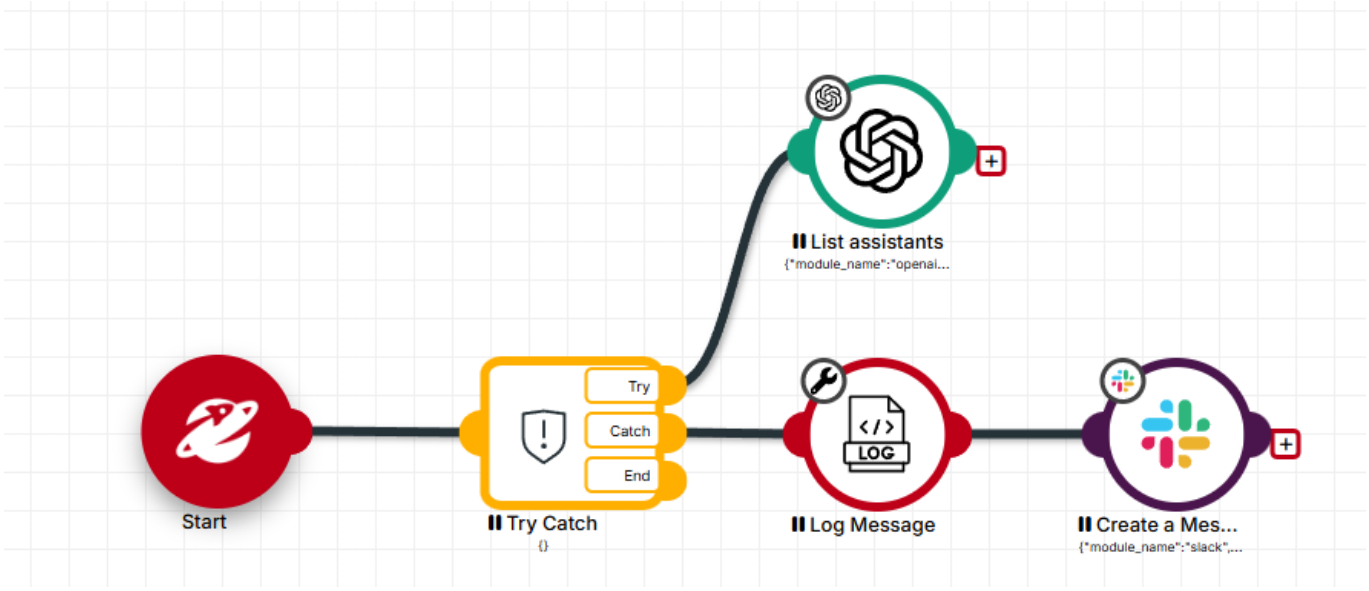


3. Manejo de Errores

Un robot en producción sin manejo de errores es una bomba de tiempo. Los errores son inevitables: APIs que fallan, bases de datos que se saturan, archivos que no llegan. Lo que distingue a un robot profesional es cómo los maneja.

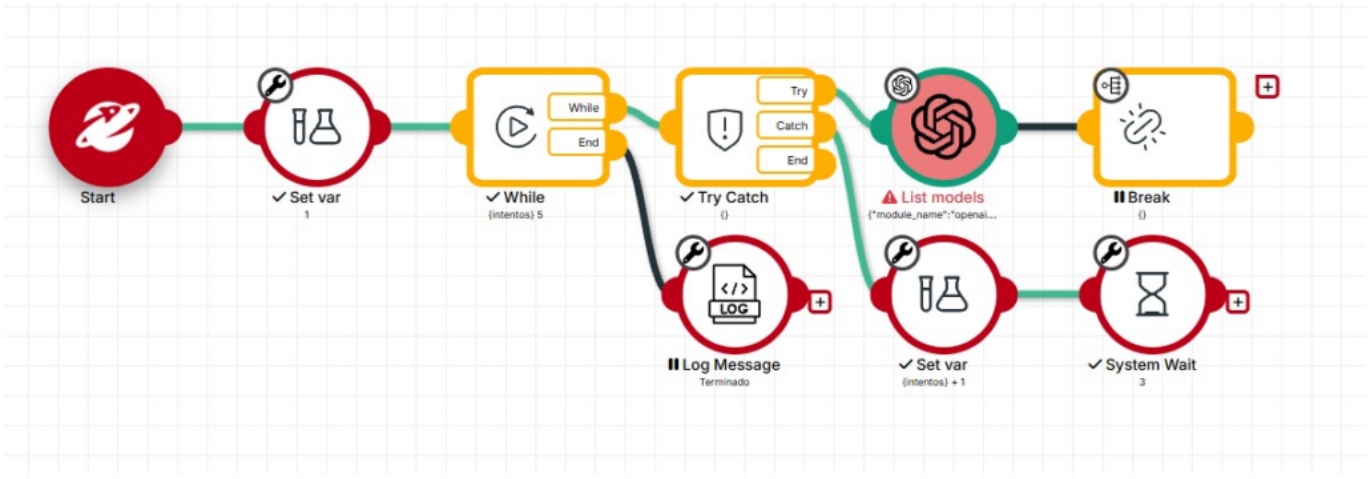
BP-06 | Todo robot de producción debe tener un bloque Try/Catch

- **Por qué:** Sin manejo de errores, cualquier falla detiene el robot silenciosamente o con un error crudo en el log. El equipo no se entera, el proceso queda a medias y el diagnóstico es difícil.
- **Cómo:** Envolver el flujo principal en un bloque Try/Catch. En el bloque Catch: 1) Loggear el error con fecha, hora y nombre del robot. 2) Enviar alerta al equipo (Teams/Discord/Gmail). 3) Terminar el flujo de forma controlada.
- **Ejemplo:** El robot VENTAS-Leads-Clasificar falla al llamar a OpenAI. Sin Catch: el proceso queda incompleto y nadie lo sabe. Con Catch: se envía un mensaje a #alertas-saturn con ‘Error en clasificación de leads a las 08:15 — Rate limit exceeded’.



BP-07 | Implementa reintentos con espera para llamadas a APIs externas

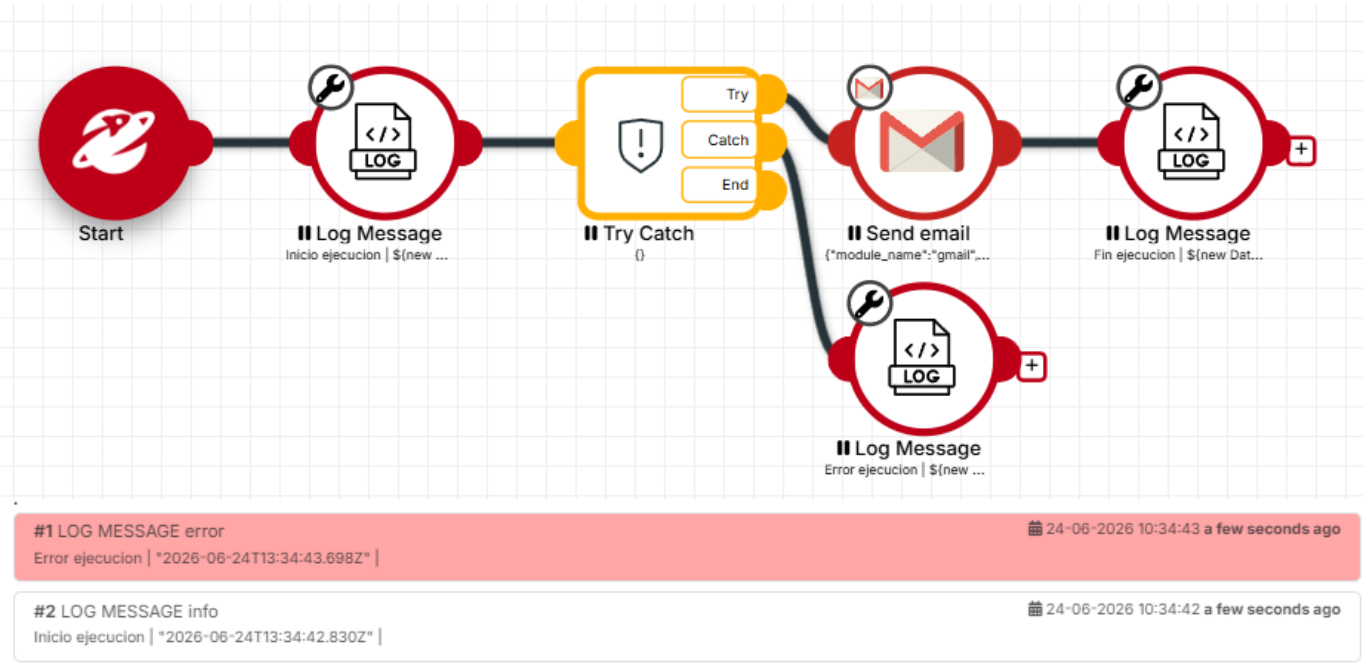
- **Por qué:** Las APIs externas (OpenAI, Gmail, Mercado Pago) pueden fallar por sobrecarga temporal, rate limits o problemas de red. Un reintento inmediato suele fallar igual. Un reintento con espera tiene alta probabilidad de éxito.
- **Cómo:** Patrón de reintento: variable reintentos = 0. Mientras reintentos < 3: intentar llamada, si falla: reintentos++, esperar (reintentos * 5) segundos, continuar loop. Si llega a 3 sin éxito: activar Catch.
- **Ejemplo:** OpenAI retorna error 429 (rate limit). El robot espera 3 segundos y reintenta. Al tercer intento exitoso, el flujo continúa normalmente. Sin este patrón, el proceso falla completamente.



BP-08 | Incluye Moons de Log estratégicas en todo el flujo

- **Por qué:** En producción no hay interfaz gráfica. El único registro de lo que pasó es el log. Sin logs, diagnosticar un fallo puede tomar días. Con logs bien ubicados, el problema es visible en minutos.
- **Cómo:** Agregar Log al inicio del flujo (datos de entrada), antes y

después de cada operación crítica (BD, API, IA) y al finalizar (resultado y métricas). Formato recomendado: [robot] [ACCION] [RESULTADO]. Incluir siempre timestamp.
- **Ejemplo:** [FIN-Facturas-Emitir] Inicio ejecucion | 2026-05-18 08:15:00 | registros_pendientes=47. Luego: [FIN-Facturas-Emitir] Factura emitida OK | id=INV-2841 | cliente=Empresa SA. Al final: [FIN-Facturas-Emitir] Fin | exito=45 | errores=2.



4. Seguridad y Credenciales

La seguridad en automatización no es opcional. Un robot con credenciales expuestas o permisos excesivos es un riesgo crítico para la empresa.

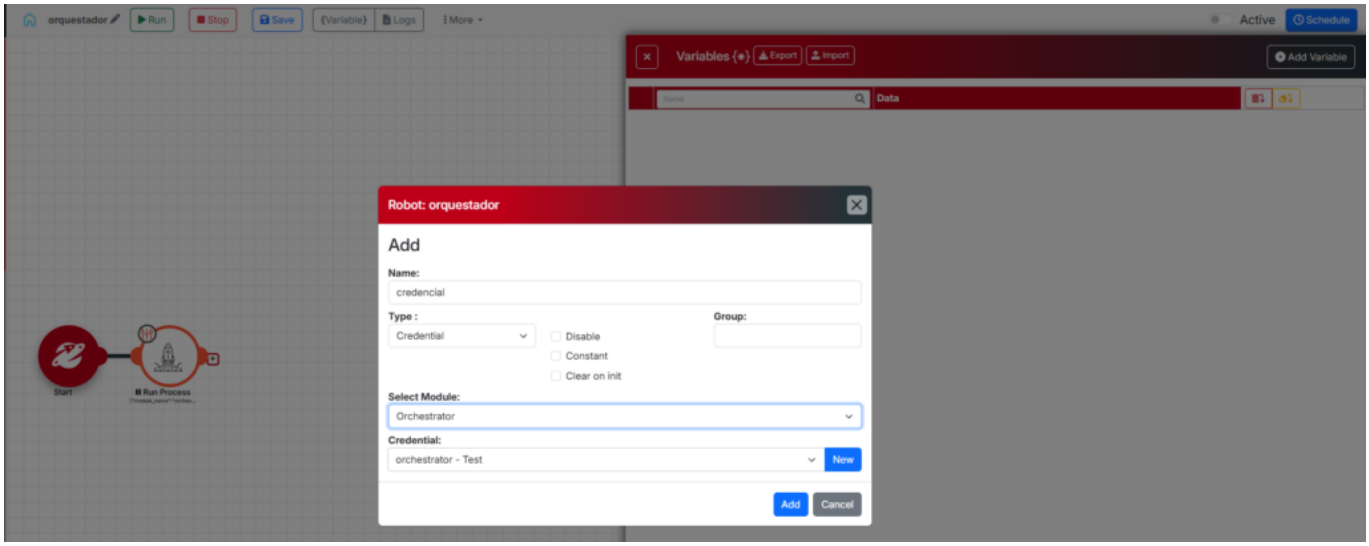
Checklist de seguridad — cumplir antes de pasar a producción

- [ ] **Nunca** escribir contraseñas, API Keys o tokens directamente en los campos de los comandos. Siempre usar credenciales del vault de Saturn Studio.
- [ ] **Nunca** compartir el archivo .json de un robot que contenga credenciales. Borrar las credenciales del robot antes de exportarlo o compartirlo.
- [ ] **Crear** usuarios de base de datos dedicados para Saturn Studio con permisos mínimos (solo las tablas y operaciones necesarias). No usar root ni sa.
- [ ] **Rotar** las API Keys cada 6 meses o inmediatamente si se sospecha una exposición. Actualizar la credencial en Saturn Studio tras la rotación.
- [ ] **Anotar** la fecha de creación de todas las API Keys del Orquestador (vigencia 2 años) en un registro interno para anticipar el vencimiento.
- [ ] **Para bases de datos:** abrir solo el puerto necesario (3306, 5432, 1433) y solo desde las IPs de Saturn Studio si el proveedor lo permite.
- [ ] **Revisar** periódicamente que las credenciales compartidas solo estén

accesibles a los usuarios que las necesitan activamente.
- [ ] **Limpiar** variables que contienen datos sensibles (tokens, contraseñas temporales) al finalizar el flujo que las utilizó.

BP-09 | Usa Variables de tipo Credential para parametrizar el acceso

- **Por qué:** Si un robot usa la credencial directa y esa credencial cambia o se necesita usar otra (ej: cambiar de ambiente dev a prod), hay que editar el robot. Con una variable de credencial, el cambio es en la variable.
- **Cómo:** En lugar de seleccionar la credencial directamente en el campo del comando, crear una Variable de tipo Credential que la referencie. En el campo del comando escribir {nombre_variable}. Para cambiar de ambiente, solo cambiar el valor de la variable.
- **Ejemplo:** Variable: openai_key (tipo Credential). En desarrollo apunta a 'OpenAI Dev'. En producción apunta a 'OpenAI Prod'. El robot es idéntico en ambos ambientes. Solo cambia la variable.



5. Rendimiento y Eficiencia

Un robot lento puede ser tan problemático como uno que falla. El rendimiento impacta directamente en el costo de las APIs, en los tiempos de respuesta al usuario y en la capacidad de procesar grandes volúmenes.

Consultas a bases de datos

✅ Hacer	❌ No hacer
Limitar el resultado de las consultas SQL con WHERE y LIMIT, especialmente en desarrollo	Hacer SELECT * sin WHERE en tablas con millones de registros

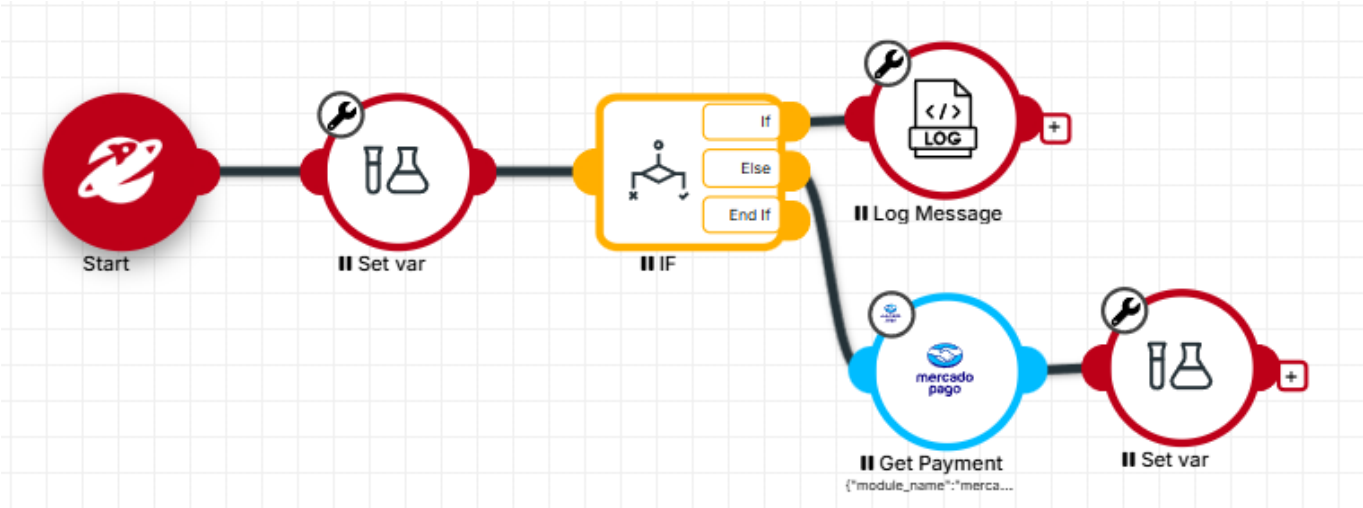
Seleccionar solo las columnas necesarias: SELECT id, nombre, estado FROM clientes	Traer todas las columnas (SELECT *) cuando solo se necesitan 2 o 3 campos
Paginar grandes volúmenes: procesar en lotes de 100-500 registros por ejecución	Intentar procesar 10.000 registros en una sola ejecución (riesgo de timeout y consumo de memoria)
Crear índices en las columnas usadas en WHERE y JOIN para acelerar las consultas	Dejar tablas sin índices y asumir que las consultas serán rápidas por defecto

Llamadas a APIs de IA

✅ Hacer	❌ No hacer
Definir max_tokens acorde al tipo de respuesta esperada (JSON simple = 200-500, análisis = 1000-2000)	Dejar max_tokens en el valor máximo por defecto en todos los comandos
Agregar una Moon Wait de 1-2 segundos entre llamadas a IA en loops para respetar rate limits	Llamar a OpenAI en loops sin espera entre iteraciones (genera errores 429 masivos)
Reutilizar el resultado de una llamada a IA si se necesita en múltiples pasos siguientes	Llamar a la misma API de IA dos veces para obtener el mismo dato (doble costo y tiempo)
Para chunks de texto: dividir documentos en bloques de 500-1000 palabras antes de enviar	Enviar documentos PDF completos de 50 páginas en un solo prompt (supera el contexto del modelo)

BP-10 | Implementa deduplicación para robots que procesan eventos externos

- **Por qué:** Un webhook puede dispararse más de una vez para el mismo evento (red inestable, reintentos del proveedor). Sin deduplicación, el robot procesa el mismo pago, correo o formulario múltiples veces.
- **Cómo:** Crear una tabla eventos_procesados en la base de datos con los campos: event_id (PK), agent_name, processed_at. Al inicio de cada ejecución: 1) Extraer el ID único del evento. 2) Consultar si ya existe. 3) Si existe: terminar. 4) Si no: procesar e insertar.
- **Ejemplo:** Un webhook de Mercado Pago puede enviar el mismo payment_id varias veces si hay timeout en la primera entrega. Sin deduplicación: se emiten 3 facturas para el mismo pago. Con deduplicación: solo se emite una.



6. Prompts de Inteligencia Artificial

La calidad del prompt determina la calidad del resultado. Un prompt mal diseñado produce respuestas inconsistentes, en formato incorrecto o con alucinaciones que rompen el flujo.

BP-11 | Define el rol, el formato de salida y las restricciones en cada prompt

- **Por qué:** Sin estas tres cosas, el modelo puede responder en cualquier formato, inventar información o dar respuestas que el flujo no puede parsear. Un prompt completo produce resultados deterministas.
- **Cómo:** Estructura recomendada del system prompt:
  1. **ROL:** “Eres un clasificador de correos de soporte técnico.”
  2. **TAREA:** Descripción precisa de lo que debe hacer.
  3. **FORMATO:** “Responde ÚNICAMENTE con un JSON válido: {"categoria": "...", "urgencia": "..."}.”
  4. **RESTRICCIONES:** “No incluyas explicaciones. No uses markdown. Si no puedes clasificar, responde {"categoria":"desconocido"}.”

- **Ejemplo:** Sin estructura: el modelo responde “Este correo parece ser una consulta sobre facturación, con urgencia media.” — imposible de parsear. Con estructura: {"categoria": "facturacion", "urgencia": "media"} — directamente usable.

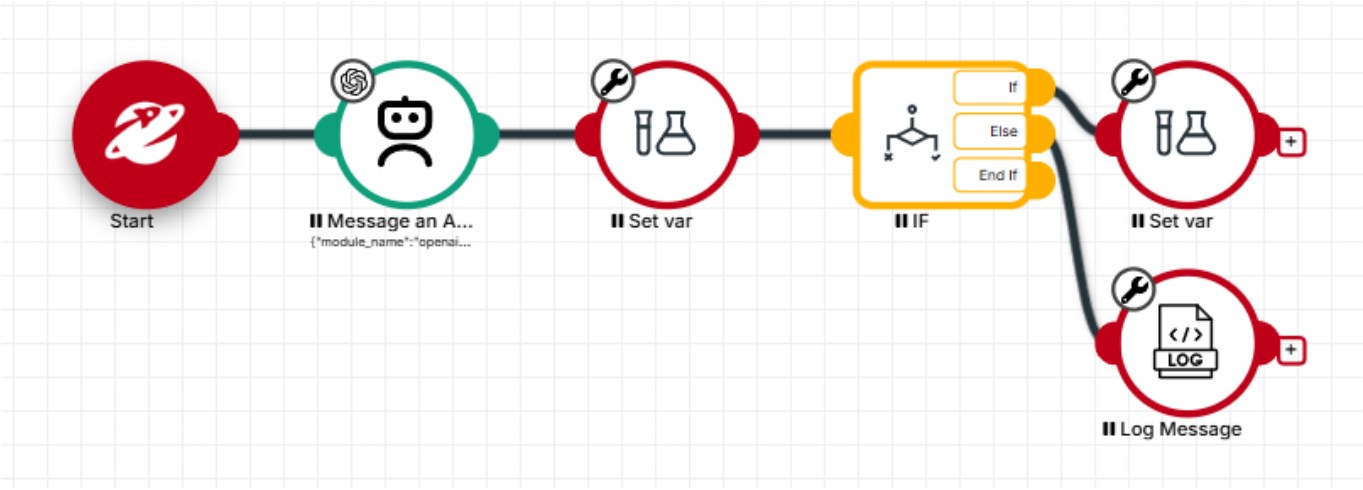
BP-12 | Incluye siempre un ejemplo del output esperado en el prompt

- **Por qué:** El *few-shot prompting* (dar un ejemplo) es la técnica más efectiva para obtener respuestas consistentes y en el formato exacto que el flujo necesita.
- **Cómo:** En el prompt agregar: EJEMPLO DE RESPUESTA CORRECTA: {"categoria":

"reclamo", "urgencia": "alta", "requiere_humano": true}. Si hay múltiples categorías posibles, listarlas explícitamente en el prompt.
- **Ejemplo:** Sin ejemplo: el modelo puede devolver categoria: reclamo, urgencia: alta (sin JSON, sin comillas). Con ejemplo: el modelo replica el formato exacto. La fiabilidad del parseo pasa del 60% al 95%.

BP-13 | Agrega una Moon de validación del JSON retornado por la IA

- **Por qué:** Incluso con un prompt perfecto, la IA puede devolver texto adicional, un JSON malformado o campos con nombres ligeramente diferentes. Un robot robusto valida antes de usar.
- **Cómo:** Después del comando de IA: 1) Limpiar el texto (remover backticks, texto previo al JSON). 2) Parsear el JSON a objeto. 3) Verificar con If/Else que los campos requeridos existen y tienen valores válidos. 4) Si la validación falla: reintentar el prompt o escalar a humano.
- **Ejemplo:** La IA retorna: ```json {"categoria": "consulta"} ```. La Moon de limpieza extrae el JSON. La validación verifica que 'categoria' existe. Si falta 'urgencia', el robot usa un valor por defecto en lugar de fallar.



7. Colaboración en Equipo

Saturn Studio está diseñado para trabajo en equipo. Estas prácticas maximizan la eficiencia colaborativa y minimizan los conflictos y la duplicación de trabajo.

BP-14 | Publica templates de robots bien probados en el Market interno

- **Por qué:** Si un robot funciona bien para un caso de uso frecuente, publicarlo como template evita que otros miembros del equipo lo reconstruyan desde cero. El Market interno es el repositorio de

conocimiento del equipo.
- **Cómo:** Al finalizar y probar un robot: 1) Limpiar las credenciales hardcodeadas. 2) Documentar el propósito en la descripción del template (max 120 caracteres). 3) Exportar como .json. 4) Subir al Market con nombre descriptivo y categoría correcta. 5) Notificar al equipo.
- **Ejemplo:** El equipo de Finanzas crea el robot FIN-Facturas-Emitir y lo publica en el Market. El equipo de Ventas lo usa como base para FIN-Cotizaciones-Emitir, ahorrando 4 horas de desarrollo.

BP-15 | Usa Variables de tipo Credential compartidas por equipo para servicios comunes

- **Por qué:** Si cada miembro del equipo crea su propia credencial de OpenAI o Gmail, cuando la clave cambia cada uno debe actualizarla por separado. Con credenciales compartidas por equipo, el cambio es centralizado.
- **Cómo:** El administrador crea la credencial 'OpenAI-Empresa' en el vault y la comparte con el equipo completo vía 'Share by Team'. Todos los robots del equipo referencian esta credencial. Si la clave rota, el admin la actualiza una vez.
- **Ejemplo:** Sin credencial compartida: 8 miembros del equipo tienen 8 credenciales de OpenAI distintas. Cuando OpenAI rota las claves, son 8 actualizaciones manuales con riesgo de que alguna quede desactualizada.

8. Paso a Producción

El paso de desarrollo a producción es el momento de mayor riesgo. Estas prácticas minimizan la probabilidad de incidentes y aceleran el diagnóstico si ocurren.

Checklist de Go-Live — completar antes de activar un robot en producción

- [ ] El robot fue probado con datos reales (no solo datos de prueba) en un ambiente de certificación o UAT.
- [ ] Todas las credenciales apuntan a cuentas de producción (no a cuentas de desarrollo o prueba).
- [ ] Las Variables Globales usan los valores de producción (DB_HOST, API_URL, etc.).
- [ ] El robot tiene manejo de errores (Try/Catch) y envía alertas al equipo si falla.
- [ ] El robot tiene Moons de Log al inicio, en puntos críticos y al final de cada ejecución.
- [ ] Se implementó deduplicación si el robot procesa eventos externos (webhooks, formularios).
- [ ] Los volúmenes de datos probados son representativos del volumen de producción (no probar con 5 registros si en producción hay 5000).

- [ ] Existe un procedimiento documentado para detener el robot si algo sale mal (quién tiene acceso, cómo hacerlo).
- [ ] Se definió quién es el responsable de monitorear el robot en las primeras 48 horas post-lanzamiento.
- [ ] Las credenciales de bases de datos usan el principio de mínimo privilegio (solo SELECT/INSERT/UPDATE en las tablas necesarias).
- [ ] El robot fue revisado por al menos otro miembro del equipo antes del go-live.

BP-16 | Implementa un ambiente de desarrollo separado del de producción

- **Por qué:** Probar cambios directamente en producción es el error más costoso en automatización. Un ambiente de desarrollo permite iterar libremente sin riesgo de afectar datos reales o procesos activos.
- **Cómo:** Usar Variables de tipo Credential para separar ambientes: variable db_connection en dev apunta a BD de prueba, en prod apunta a BD real. Variable api_url en dev apunta a sandbox del servicio, en prod al endpoint real. Cambiar de ambiente = cambiar el valor de la variable.
- **Ejemplo:** El robot FIN-Facturas-Emitir en dev apunta a la API de pruebas de Nubox (UAT) y a una base de datos de test. En prod apunta a la API real y la BD de producción. El flujo es idéntico, solo cambian las variables.

BP-17 | Monitorea las primeras ejecuciones en producción activamente

- **Por qué:** Las primeras ejecuciones en producción siempre revelan casos bordes que no aparecieron en pruebas: datos en formatos inesperados, volúmenes mayores al estimado, interacciones con sistemas que se comportan diferente.
- **Cómo:** Durante las primeras 48 horas post-lanzamiento: 1) Revisar el log de cada ejecución manualmente. 2) Comparar los resultados con el proceso manual que se estaba haciendo antes. 3) Tener el robot pausado si se detecta cualquier anomalía. 4) Documentar los casos bordes encontrados para ajustar el robot.
- **Ejemplo:** El robot de clasificación de correos funciona perfectamente en pruebas. En producción aparecen correos en otro idioma (no esperado) que la IA no puede clasificar. Sin monitoreo activo: se acumulan sin procesar. Con monitoreo: se detecta el día 1 y se agrega un caso al prompt.

Resumen — Las 17 Mejores Prácticas

Código	Categoría	Práctica
BP-01	Nomenclatura	Nombra los robots con patrón [AREA]-[PROCESO]-[ACCION]
BP-02	Documentación	Documenta cada Moon con una descripción breve de su propósito
BP-03	Organización	Organiza los robots en carpetas por área y estado
BP-04	Variables	Valida el resultado de cada Moon crítica antes de usarlo
BP-05	Variables	Usa Variables Globales para configuración compartida entre robots
BP-06	Errores	Todo robot de producción debe tener un bloque Try/Catch
BP-07	Errores	Implementa reintentos con espera para llamadas a APIs externas
BP-08	Errores	Incluye Moons de Log estratégicas con timestamp en todo el flujo
BP-09	Seguridad	Usa Variables de tipo Credential para parametrizar el acceso por ambiente
BP-10	Rendimiento	Implementa deduplicacion para robots que procesan eventos externos
BP-11	Prompts IA	Define rol, formato de salida y restricciones en cada prompt
BP-12	Prompts IA	Incluye siempre un ejemplo del output esperado (*few-shot prompting*)
BP-13	Prompts IA	Agrega una Moon de validación del JSON retornado por la IA
BP-14	Equipo	Publica templates de robots bien probados en el Market interno
BP-15	Equipo	Usa credenciales compartidas por equipo para servicios comunes
BP-16	Producción	Implementa un ambiente de desarrollo separado del de producción
BP-17	Producción	Monitorea activamente las primeras 48 horas post-lanzamiento