Centro de Enseñanza Técnica Industrial

Desarrollo de Software

Proyecto Segundo Parcial - Storix

Jesús Alberto Aréchiga Carrillo

22310439 6N

Profesor

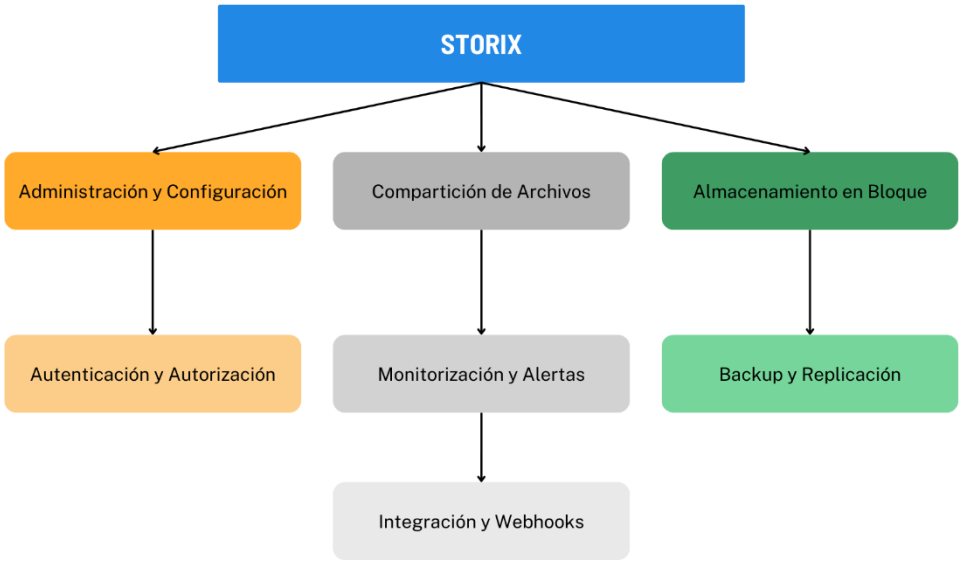
José Francisco Pérez Reyes

Marzo 2025

Guadalajara, Jalisco

Introducción

Storix constituye una plataforma integral de almacenamiento basada en una arquitectura de microservicios, diseñada para proporcionar capacidad de expansión, resiliencia y estabilidad. El presente documento detalla el proceso completo de desarrollo del servicio, abarcando desde la identificación de requerimientos hasta la operación y la mejora continua, e integrando los principios de ITILv4 en cada fase. El objetivo es evidenciar, de forma clara y accesible para los altos directivos, la solidez y eficacia de Storix en la gestión de servicios.



Identificación de Requerimientos y Estrategia del Servicio

La fase inicial del proceso se centra en la identificación de las necesidades y la definición de la estrategia del servicio. En esta etapa se recopilan las demandas de los usuarios y se alinean con los objetivos estratégicos de la organización, determinándose el alcance y los recursos necesarios para la solución. Se evalúa la situación actual y se proyectan las necesidades futuras, de manera que el servicio de almacenamiento se ajuste de forma coherente a las metas empresariales. Esta etapa se enmarca en la fase de Service Strategy de ITILv4, la cual orienta la formulación de propuestas de valor y establece las bases para un servicio que contribuya de manera sostenida al éxito de la organización.

Diseño y Planificación del Servicio

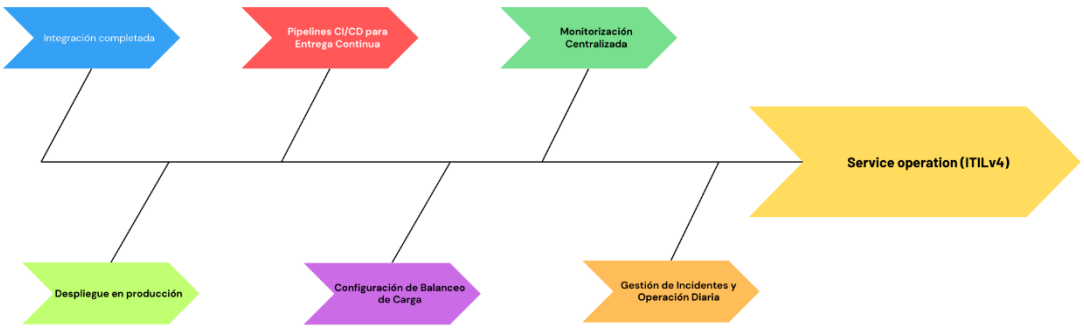
Una vez definidos los requerimientos, se procede al diseño y la planificación del servicio. Durante esta fase, se estructura Storix como un conjunto de componentes modulares, en los que cada microservicio cumple una función específica, desde la administración hasta la integración con sistemas externos. Se establecen normas de escalabilidad, continuidad, calidad y seguridad, así como la planificación de procesos de integración y despliegue continuo (CI/CD). Este enfoque, acorde con la fase de Service Design de ITILv4, garantiza que todos los componentes sean desarrollados bajo altos estándares de calidad y estén preparados para adaptarse a futuros cambios.

Desarrollo de Integración del Servicio

Con el diseño establecido, se procede al desarrollo e integración de los microservicios. En esta etapa, cada componente se implementa de forma independiente y se somete a rigurosas pruebas unitarias, de integración y de rendimiento, asegurando su correcto funcionamiento tanto de forma individual como en conjunto. La integración se efectúa mediante el uso de APIs y webhooks, apoyándose en tecnologías de contenedores y orquestadores, como Kubernetes, para facilitar una implementación ágil y escalable. Esta fase se corresponde con el proceso de Service Transition de ITILv4, que gestiona los cambios de manera controlada y asegura una transición ordenada hacia el entorno operativo.

Implementación y Operación

Una vez integrados los componentes, se procede al despliegue de Storix en el entorno productivo. La implementación se lleva a cabo mediante pipelines CI/CD que garantizan una entrega continua y controlada, complementadas con mecanismos de balanceo de carga, redundancia y tolerancia a fallos para asegurar la continuidad del servicio. Durante la operación diaria, se emplea un sistema centralizado de monitorización que recoge métricas, registros y notificaciones en tiempo real, permitiendo una respuesta inmediata ante cualquier incidencia. Esta etapa se enmarca en el ámbito de Service Operation de ITILv4, el cual define los procesos y herramientas necesarios para mantener la estabilidad, seguridad y eficiencia operativa del servicio.



Mejora Continua

La fase final se centra en la mejora continua, en la que se evalúa el rendimiento del servicio y se implementan ajustes basados en datos empíricos y la retroalimentación de los usuarios. Se analizan indicadores de desempeño (SLA/SLO) y se identifican oportunidades de optimización que permitan adaptar el servicio a nuevas demandas y desafíos. Este enfoque se fundamenta en el proceso de Continual Service Improvement (CSI) de ITILv4, que promueve revisiones periódicas y el uso de indicadores clave (KPIs) para asegurar que Storix continúe evolucionando y ofreciendo un valor sostenido a la organización.

Sistema de Valor del Servicio (SVS)

El Sistema de Valor del Servicio (Service Value System, SVS) de ITIL 4 establece el marco global mediante el cual una organización crea, entrega y mejora continuamente servicios que generan valor para sus clientes y partes interesadas. Storix se inserta en este SVS al articular claramente sus flujos de valor (value streams) y prácticas recolectadas en la plataforma, de tal manera que cada microservicio —Administración y Configuración, Compartición de Archivos, Almacenamiento en Bloque, Autenticación y Autorización, Monitorización y Alertas, Backup y Replicación, e Integración y Webhooks— contribuye de forma integrada a la cadena de valor del servicio.

En el SVS de Storix, los flujos de valor mapean las actividades necesarias para responder a características específicas de los clientes, tales como la provisión de nuevos volúmenes, la atención de incidentes críticos o el cumplimiento de solicitudes de acceso. Cada flujo de valor se apoya en prácticas de gestión (por ejemplo, Control de Cambios, Gestión de Incidentes, Gestión de Solicitudes) y en prácticas técnicas (por ejemplo, Gestión de Implementación, Administración de Infraestructura), asegurando que todas las actividades estén coordinadas y controladas.

Para garantizar una visión holística, ITIL 4 propone cuatro dimensiones que deben estar equilibradas y alineadas:

1. **Organización y Personas**

   Storix define roles claros —Solicitante, Equipo de Evaluación de Riesgos, Autoridad de Cambio, Coordinador de Cambios, Equipo de Implementación, Revisor Post-implementación, Mesa de Servicio, Soporte Técnico y Proveedores externos— junto con responsabilidades y competencias necesarias. El modelo de gobernanza y las políticas de ITIL

v4 aseguran que exista una estructura organizativa que respalde la agilidad y la responsabilidad en cada práctica.

2. **Información y Tecnología**

   La capacidad de Storix para gestionar grandes volúmenes de datos y orquestar microservicios depende de un repositorio de configuración centralizado (CMDB), de herramientas de automatización (CI/CD, orquestadores como Kubernetes) y de sistemas de monitorización (por ejemplo, Prometheus, Grafana). Los flujos de datos, los registros de auditoría y los dashboards de métricas son fuentes primordiales para la toma de decisiones y la mejora continua.

3. **Socios y Proveedores**

   Storix interactúa con fabricantes de hardware, servicios de nube pública, proveedores de software (por ejemplo, plataformas de autorización externa, integraciones de notificaciones) y consultores de ciberseguridad. El modelo de gestión de proveedores de ITIL v4 asegura contratos claros, SLAs acordados y puntos de escalación definidos para mantener la calidad del servicio.

4. **Valor y Procesos**

   Cada microservicio de Storix genera un conjunto de outputs medibles (nuevos recursos provisionados, incidentes resueltos, backups exitosos, etc.) que se traducen en valor para el cliente. Los procesos definidos —desde la Gestión de Solicitudes de Servicio hasta la Mejora Continua— garantizan que el valor no solo se entregue, sino que también se optimice y evolucione según las necesidades cambiantes del negocio.

Al integrar el SVS y sus cuatro dimensiones en el diseño de Storix, se establece una base sólida para alinear la estrategia de TI con los objetivos corporativos, garantizando consistencia, transparencia y un enfoque centrado en el cliente en todas las prácticas y actividades de la plataforma.

Práctica de mejora continua

Modelo de Mejora en Siete Pasos

1. **Visión y alcance**

   Se establecen los objetivos estratégicos de mejora —por ejemplo, reducir el MTTR (Mean Time to Restore) en un 20% o incrementar la disponibilidad de snapshots al 99,9%— vinculándolos con los indicadores de rendimiento (KPIs) y los acuerdos de nivel de servicio (SLA).

2. **¿Dónde nos encontramos?**

   A través de dashboards de monitorización, registros de incidentes y métricas de uso (por ejemplo, volumen de solicitudes de provisionamiento, tasa de éxito de backups), se genera un mapa preciso del desempeño de Storix en el periodo de análisis.

3. **¿Dónde queremos estar?**

   Se fijan metas concretas y alcanzables basadas en benchmarks internos o estándares de la industria, como tiempos máximos de respuesta de 4 horas para solicitudes de información o un índice de satisfacción de usuarios superior al 90 %.

4. **¿Cómo llegaremos allí?**

   Se priorizan iniciativas (por ejemplo, automatización de workflows de solicitud, optimización de índices en la base de datos de configuración, ampliación de la capacidad de autoescalado) y se asignan recursos, responsables, cronogramas y criterios de éxito.

5. **Tomar acción**

   Bajo procesos de Control de Cambios, las mejoras se despliegan mediante pipelines CI/CD en entornos de staging y producción, asegurando pruebas previas y reversión rápida en caso de desviaciones.

6. **¿Llegamos allí?**

   Tras la implementación, se comparan las métricas post-cambio con los valores de referencia, evaluando indicadores como el MTTR, la tasa de resolución en primer contacto y la disminución de incidentes recurrentes.

7. **Mantener el Impulso y Retroalimentar**

   La retroalimentación de usuarios (encuestas de satisfacción) y de los equipos de soporte alimenta nuevos ciclos de CSI, refinando políticas, procedimientos y automatizaciones hasta alcanzar niveles óptimos de calidad y eficiencia.

Aplicación Práctica en Storix

- **Recolección Continua de Datos:** Los microservicios de Monitorización y Alertas recopilan métricas en tiempo real y almacenan históricos en la CMDB para alimentar análisis de tendencias.
- **Paneles de Control CSI:** Se disponen dashboards específicos para la práctica de Mejora Continua, con indicadores visuales de cumplimiento de objetivos y alertas tempranas cuando se desvían de los umbrales definidos.

- **Reuniones de Revisión CSI:** Se convocan revisiones trimestrales con representantes de Operaciones, Desarrollo, Seguridad y Negocio para validar progresos, reordenar prioridades y aprobar nuevos proyectos de optimización.
- **Documentación de Lecciones Aprendidas:** Cada iniciativa de mejora se documenta en un repositorio de conocimiento compartido, facilitando la replicación de éxitos y la prevención de errores previos en futuros ciclos.
- **Integración con el Catálogo de Servicios:** Las mejoras aprobadas se reflejan en actualizaciones del Catálogo de Servicios y en los procedimientos de solicitud, garantizando que los usuarios accedan siempre a la versión más eficiente de Storix.

Control de cambios

Roles y responsabilidades

Solicitante: Inicia la solicitud de cambio aportando la justificación, el alcance y los recursos necesarios.

Equipo de Evaluación de Riesgos: Especialistas en seguridad, operaciones y arquitectura que identifican y valoran riesgos y beneficios.

Autoridad de Cambio: Autoriza o rechaza la solicitud de cambio según los resultados de la evaluación de riesgos y el impacto.

Coordinador de Cambios: Gestiona el cronograma, comunica el plan y vela por la correcta documentación y cierre del cambio.

Equipo de Implementación: Ejecuta el cambio siguiendo el plan aprobado y documenta cualquier desviación.

Revisor Post-implementación: Verifica el logro de los objetivos, registra lecciones aprendidas y cierra la solicitud de cambio.

Procedimiento

Presentación y Registro

1. El Solicitante completa una solicitud de cambio que incluya descripción, justificación, alcance y recursos.
2. El Coordinador de Cambios registra la solicitud de cambio en la herramienta de gestión y asigna un identificador único.

Identificación y Evaluación de Riesgos

1. El Equipo de Evaluación de Riesgos revisa la solicitud de cambio para identificar riesgos técnicos, de seguridad, de negocio y de continuidad.
2. Se asigna una calificación de riesgo (bajo, medio, alto) y se documentan las contramedidas recomendadas.

Revisión de Impacto

1. Se analiza el impacto potencial en disponibilidad de servicio, experiencia de usuario y otros elementos dependientes.
2. Se actualiza la solicitud de cambio con los resultados y se definen las ventanas de mantenimiento propuestas.

Aprobación

Según el tipo de cambio:

- Estándar: preaprobado y programable inmediatamente.
- Normal: la Autoridad de Cambio revisa riesgos e impactos y decide.
- Emergencia: autorización acelerada por responsable designado.

El Coordinador de Cambios notifica la decisión al equipo de implementación.

Programación y Comunicación

1. El Coordinador de Cambios inserta el cambio aprobado en el Cronograma de Cambios.
2. Se envía un comunicado a los stakeholders con fecha, hora, alcance y plan de contingencia.

Implementación

1. El Equipo de Implementación ejecuta el cambio siguiendo el plan aprobado.
2. Se documentan incidencias, desviaciones y tiempos reales de ejecución.

Revisión Post-Implementación

1. El Revisor Post-implementación verifica el cumplimiento de objetivos y busca efectos adversos.
2. Se registra un informe final con lecciones aprendidas y la solicitud de cambio se cierra formalmente.

Flujos de autoridad de cambio

Cambios Estándar: Autorizados automáticamente bajo criterios predefinidos.
Cambios Normales: Evaluación por Comité Técnico; autorización del Gerente de Operaciones.
Cambios de Emergencia: Evaluación y autorización por el Equipo de Respuesta a Incidentes.

Herramientas y documentación

Se utilizan plataforma ITSM para seguimiento de la solicitud de cambio, plantillas de evaluación de riesgos e impactos, cronograma de cambios integrado y reportes de cierre con lecciones aprendidas.

Gestión de incidentes

La práctica de Gestión de Incidentes tiene como objetivo principal restaurar lo más rápido posible el funcionamiento normal de los servicios tras una interrupción no planificada o una degradación en su calidad, minimizando así el impacto negativo en el negocio y sus usuarios. En el contexto de Storix, esta práctica se implementa a través de un proceso claramente definido que cubre desde el registro inicial hasta el cierre formal de cada incidente, garantizando que se cumplan los Acuerdos de Nivel de Servicio (SLA) y se preserve la satisfacción del cliente.

Definición y Alcance

Un incidente se define como cualquier interrupción no planificada de un servicio o reducción en su calidad. Quedan excluidos de esta categoría los cambios autorizados y las solicitudes de servicio predefinidas. Todos los incidentes que afecten a los microservicios de Storix —ya sean fallos en la provisión de almacenamiento, en la autenticación o en la replicación de datos— deben registrarse y gestionarse conforme a este procedimiento.

Ciclo de Vida del Incidente

1. **Detección y Registro**
    - El incidente puede originarse por una alerta automatizada del microservicio de Monitorización y Alertas o por reportes de usuarios/mesa de servicio.
    - Se genera un ticket con un identificador único, fecha/hora de apertura y descripción inicial detallada.
2. **Clasificación y Priorización**
    - Cada incidente se clasifica según el servicio afectado (p. ej., Compartición de Archivos, Almacenamiento en Bloque) y se asigna una prioridad basada en su **Impacto** (número de usuarios o procesos afectados) y **Urgencia** (gravedad del efecto).
    - Se utiliza una matriz de prioridad predefinida para garantizar consistencia y rapidez en la toma de decisiones.
3. **Asignación y Diagnóstico Inicial**
    - La mesa de servicio asigna el ticket al equipo de soporte adecuado (operaciones, desarrollo o proveedor) y, de ser necesario, activa un **equipo de enjambre** para incidentes mayores.
    - Se recogen datos de logs, métricas de desempeño y pasos de reproducción para identificar la causa raíz provisional.
4. **Resolución y Recuperación**

    - El equipo responsable aplica la solución —sea un parche, un cambio de configuración o un workaround documentado— procurando restaurar el servicio dentro del **tiempo objetivo acordado**.
    - Para incidentes críticos, se invocan planes de recuperación ante desastres o escalación a especialistas externos.
5. **Cierre del Incidente**
    - Una vez verificado el restablecimiento del servicio y la satisfacción del usuario, se documentan las acciones realizadas, el tiempo total de resolución y las lecciones aprendidas.
    - El ticket se cierra formalmente y se actualizan los registros de incidentes para futuros análisis.

Roles y Responsabilidades

- **Usuario/Mesa de Servicio:** Detecta y reporta el incidente, proporciona información contextual.
- **Analista de Primera Línea:** Realiza el diagnóstico inicial, aplica soluciones predefinidas y evalúa la necesidad de escalación.
- **Equipo de Soporte Especializado:** Gestiona la resolución de incidentes complejos o de alto impacto, coordina el equipo de enjambre si procede.
- **Coordinador de Incidentes Mayores:** Activa planes de contingencia y asegura la comunicación entre todos los stakeholders.
- **Revisor de Cierre:** Verifica que se hayan cumplido los criterios de cierre y que el usuario confirme la restauración satisfactoria.

Métricas y SLA

- **MTTR (Mean Time to Restore):** Tiempo promedio desde la detección hasta la restauración del servicio.
- **Tasa de Resolución en Primer Contacto:** Porcentaje de incidentes resueltos por el analista de primera línea sin escalación.
- **Porcentaje de Incidentes Cerrados Dentro del SLA:** Proporción de tickets resueltos antes del tiempo objetivo acordado.
- **Número de Incidentes Recurrentes:** Indicador de problemas subyacentes que requieren acciones de mejora o gestión de problemas.

Integración con la Cadena de Valor

- **Entregar y Asistir:** La gestión de incidentes es la práctica central para mantener la continuidad operativa y garantizar la disponibilidad del servicio.

- **Involucrar:** Asegura comunicación constante y transparente con los usuarios y patrocinadores.
- **Mejorar:** Los registros de incidentes alimentan la práctica de Mejora Continua, identificando áreas de optimización y recurrentes.
- **Diseñar y Transición:** Los incidentes detectados durante despliegues o pruebas previas informan ajustes en futuros diseños y procesos de transición.

Formato de Registro de Incidentes

ID	Fecha/Hora	Servicio Afectado	Descripción	Impacto	Prioridad	Responsable	Estado

Gestión de problemas

La práctica de Gestión de Problemas busca reducir la frecuencia y el impacto de los incidentes al identificar y abordar de manera proactiva sus causas raíz, ya sean conocidas o potenciales. Para Storix, esto implica una aproximación sistemática que va más allá de la simple resolución puntual de incidentes, garantizando que las fallas subyacentes se investiguen, documenten y solucionen, o que se apliquen soluciones temporales (workarounds) mientras se desarrolla una corrección definitiva.

Definiciones Clave

- **Problema:** Causa real o potencial de uno o más incidentes.
- **Error Conocido:** Problema analizado cuyo origen se ha identificado, pero para el cual aún no existe una solución definitiva.
- **Workaround (Solución Alternativa):** Medida temporal que reduce o elimina el impacto de un incidente o problema, documentada hasta que exista una corrección permanente.

Fases del Ciclo de Gestión de Problemas

1. **Identificación del Problema**
    - Análisis de tendencias en los registros de incidentes para detectar patrones y recurrencias.
    - Detección de duplicados y quejas recurrentes provenientes de usuarios, mesa de servicio y equipos de soporte.

- Durante la gestión de incidentes mayores, evaluación del riesgo de repetición y recopilación de aportes de proveedores, desarrolladores, equipos de prueba y proyectos.

2. **Control del Problema**
    - Priorización de los problemas identificados en función de su impacto potencial y probabilidad de ocurrencia, enfocando recursos primero en los de mayor riesgo.
    - Investigación de causas múltiples y análisis en profundidad, considerando las cuatro dimensiones del servicio (organización, información, socios y tecnología).
    - Documentación de diagnósticos y de cualquier solución temporal aplicada, con definición de criterios de efectividad y plan de seguimiento.
3. **Control del Error**
    - Transición de un problema a “Error Conocido” cuando se disponga de un workaround estable o cuando la solución completa no sea viable de inmediato.
    - Mantenimiento de un registro de errores conocidos que incluya la descripción de la causa raíz, la solución alternativa y el estado de la corrección definitiva.
    - Evaluación periódica de la eficacia de los workarounds y actualización de la documentación con mejoras o nuevas observaciones.

Roles y Responsabilidades

- **Analista de Problemas:** Lidera la identificación, el análisis de raíces y la elaboración de workarounds, coordinando con proveedores y equipos técnicos.
- **Propietario del Problema:** Responsable de supervisar el ciclo completo, desde la detección hasta la verificación de la solución permanente.
- **Equipo de Gestión de Errores:** Define criterios para mover problemas a estado de error conocido, mantiene la base de datos de errores y revisa la efectividad de las soluciones alternativas.
- **Stakeholders Técnicos y de Negocio:** Aportan información contextual, validan prioridades y colaboran en la definición de planes de corrección.

Métricas y Reportes

- **Número de Problemas Identificados:** Total de causas raíz detectadas en un periodo.

- **Tiempo Promedio de Análisis de Problemas:** Intervalo desde la identificación inicial hasta la definición de la causa raíz.
- **Porcentaje de Workarounds Efectivos:** Proporción de soluciones alternativas que resuelven satisfactoriamente el incidente sin generar efectos secundarios.
- **Número de Errores Conocidos Pendientes:** Problemas categorizados como errores conocidos sin corrección definitiva.
- **Reducción de Incidentes Recurrentes:** Disminución proporcional de incidentes asociados a problemas ya identificados.

Integración con la Cadena de Valor

- **Obtener/Construir:** La detección y análisis de problemas en entornos de desarrollo permiten corregir defectos antes del lanzamiento.
- **Diseñar y Transición:** Los hallazgos derivados de problemas informan ajustes en los procesos de prueba y despliegue.
- **Entregar y Asistir:** La documentación de workarounds mejora la velocidad de resolución en incidentes futuros.
- **Mejorar:** El análisis continuo de problemas alimenta la práctica de Mejora Continua con iniciativas de optimización y prevención.

Formato de Registro de Problemas

ID Problema	Fecha Identificación	Descripción	Causa Raíz	Solución Alternativa	Estado

Gestión de solicitudes de servicio

Definición y Alcance

Una **solicitud de servicio** es cualquier petición iniciada por un usuario o un representante autorizado que forma parte del catálogo de servicios acordado (por ejemplo: reposición de consumibles, provisión de espacio de almacenamiento, creación de cuentas de acceso o consultas de información). Estas solicitudes no representan fallos del servicio, sino actividades estándar, muchas de ellas preautorizadas (cambios estándar), que contribuyen a las fases de “Obtener/Construir” y “Diseñar/Transición” del SVS de ITIL 4.

Tipos de Solicitud

- **Provisión de Servicio o Recurso:** Asignación de almacenamiento, cuentas de usuario, licencias de software.

- **Acceso a Servicios:** Alta, modificación o revocación de permisos en aplicaciones corporativas.
- **Entrega de Productos o Consumibles:** Suministro de cartuchos de tóner, medios de almacenamiento físico o informes predefinidos.
- **Solicitudes de Información:** Peticiones de reportes, estadísticas o documentación técnica.
- **Feedback y Quejas:** Felicitaciones, sugerencias de mejora o reclamaciones relacionadas con el servicio.

Proceso de Gestión de Solicitudes

1. **Registro y Clasificación**
    - El usuario completa un formulario en el portal de autoservicio, seleccionando el tipo de solicitud y proporcionando detalles.
    - La **Mesa de Servicio** verifica la validez y categoriza la solicitud según el catálogo, asignándole un nivel de prioridad y un SLA objetivo.
2. **Aprobación (si aplica)**
    - Para solicitudes que requieren autorización (por ejemplo, aprovisionamiento con coste o acceso sensible), la solicitud se remite al **Autoridad de Solicitud** definida (jefatura, Finanzas, Seguridad).
    - Las solicitudes estándar sin impacto o coste predefinido se consideran preautorizadas y avanzan directamente al cumplimiento.
3. **Cumplimiento**
    - El **Equipo de Cumplimiento** sigue un flujo de trabajo documentado y validado, apoyado por scripts y automatizaciones, para llevar a cabo la solicitud.
    - Se emplean pipelines CI/CD y herramientas de gestión de configuración para garantizar consistencia en entornos de desarrollo, prueba y producción.
4. **Notificación y Cierre**
    - Una vez completado el servicio, el usuario recibe una notificación automática con la confirmación y, si procede, un enlace a la encuesta de satisfacción.
    - La solicitud se marca como “Completada” en el sistema ITSM y se documentan posibles desviaciones o tiempos reales de cumplimiento.

Roles y Responsabilidades

- **Usuario/Representante Autorizado:** Inicia la solicitud mediante el portal de autoservicio.
- **Mesa de Servicio:** Registra, clasifica y da seguimiento hasta la asignación al equipo de cumplimiento.
- **Autoridad de Solicitud:** Valida y autoriza aquellas solicitudes que afectan políticas financieras, de seguridad o de licenciamiento.
- **Equipo de Cumplimiento:** Ejecuta las tareas necesarias según el proceso definido, documenta resultados y respeta el SLA acordado.
- **Coordinador de Solicitudes:** Monitorea indicadores de cumplimiento, gestiona el catálogo de servicios y promueve mejoras de proceso.

Métricas y SLAs

- **Tiempo de Cumplimiento Promedio:** Intervalo medio desde la solicitud hasta su finalización.
- **Porcentaje de Solicitudes Cumplidas Dentro del SLA:** Proporción de peticiones atendidas en el plazo objetivo.
- **Tasa de Automatización de Solicitudes:** Porcentaje de solicitudes gestionadas de forma completamente automática.
- **Satisfacción del Usuario:** Valoración media obtenida a través de la encuesta de satisfacción tras el cierre de la solicitud.

Integración con la Cadena de Valor

- **Obtener/Construir:** Gestiona la adquisición y provisionamiento de componentes y recursos preaprobados.
- **Diseñar/Transición:** Ejecuta cambios estándar durante despliegues y puesta en marcha de nuevos servicios.
- **Entregar y Asistir:** Constituye el canal principal de interacción con el usuario para actividades cotidianas.
- **Mejorar:** El feedback y las métricas de cumplimiento alimentan iniciativas de mejora continua, optimizando procesos y ampliando la automatización.

Catálogo de servicios y Mesa de servicio

El Catálogo de Servicios es un repositorio estructurado y accesible que describe de forma detallada los servicios acordados con los usuarios, sus niveles de desempeño y los procedimientos de solicitud. Para Storix, el catálogo incluye:

- **Nombre y Descripción**

Cada servicio se presenta con un título claro y un resumen de su funcionalidad y alcance.

- **Procedimiento de Solicitud**

  Pasos estandarizados para que el usuario inicie la acción (portal de autoservicio, formulario específico, datos requeridos).

- **Tipo de Solicitud y Clasificación**

  Identifica si es un servicio de información, provisión, acceso, entrega física o cambio estándar, para determinar el flujo de trabajo interno.

- **Requisitos de Aprobación**

  Indica cuándo se necesita autorización (seguridad, finanzas, compras) y a qué nivel.

- **Acuerdos de Nivel de Servicio (SLA)**

  Tiempo objetivo de cumplimiento, horario de atención y canales de soporte.

- **Indicadores de Calidad**

  Métricas específicas (porcentaje de solicitudes automatizadas, satisfacción del usuario, tiempo promedio de respuesta) que permiten a los responsables monitorear y mejorar cada servicio.

Este catálogo se mantiene actualizado dentro de la herramienta ITSM y se expone a los usuarios vía portal web, garantizando transparencia y autoayuda.

Mesa de Servicio

La Mesa de Servicio es el punto único de contacto para todos los usuarios de Storix, encargado de recibir, clasificar y canalizar:

1. **Incidentes:**
   - Registro inicial, diagnóstico de primer nivel y resolución de aquéllos con baja complejidad, o escalación al equipo especializado.
2. **Solicitudes de Servicio:**
   - Validación de la petición según el Catálogo de Servicios, clasificación y, si procede, autorización o derivación al Equipo de Cumplimiento.
3. **Información y Feedback:**
   - Atención de consultas generales, gestión de felicitaciones y quejas, y recopilación de sugerencias para alimentar la mejora continua.

Funciones Clave de la Mesa de Servicio

- **Comunicación Única:** Mantener informados a los usuarios sobre el estado de sus tickets y cambios programados (manteniendo el cronograma de cambios accesible).
- **Coordinación de Recursos:** Asignar técnicos, proveedores o equipos de enjambre según la naturaleza y prioridad de cada caso.
- **Escalación y Seguimiento:** Aplicar criterios de prioridad basados en el Impacto/Urgencia, garantizando que los casos críticos se atiendan bajo planes de contingencia claros.
- **Base de Conocimiento:** Documentar soluciones frecuentes, workarounds y FAQs para empoderar a los usuarios y reducir la carga de trabajo repetitiva.
-

Integración con la Cadena de Valor

- **Involucrar:** Facilita la comunicación continua con clientes y usuarios, recopilando feedback y coordinando las expectativas de servicio.
- **Diseñar y Transición:** Sirve como nexo durante despliegues, informando a los usuarios de cambios y gestionando los tickets generados en entornos de prueba.
- **Entregar y Asistir:** Constituye la práctica central para la atención de incidentes y solicitudes, garantizando la entrega normal del servicio.
- **Mejorar:** Los datos recopilados (volumen y tipo de tickets, tiempo de resolución, satisfacción) alimentan las iniciativas de mejora continua, optimizando el catálogo y la operativa de la mesa.

Gestión del nivel de servicio

La práctica de Gestión del Nivel de Servicio asegura que los niveles de servicio entregados por Storix se definan, acuerden, supervisen y revisen de forma continua, alineándose con las expectativas del negocio y los usuarios. Esta gestión se fundamenta en acuerdos documentados (Service Level Agreements, SLA) y métricas claras que permiten medir objetivamente el desempeño y garantizar la mejora permanente.

Definición de Acuerdos de Nivel de Servicio

Para cada servicio del catálogo se establecen:

- Objetivos de Disponibilidad
- Tiempos de Respuesta y Resolución

- Ventanas de Mantenimiento programadas y excepciones, comunicadas con antelación para minimizar impactos.

Los SLA se documentan en contratos internos y externos, y se comunican al personal de soporte y a los usuarios finales.

Diseño de Métricas y KPIs

- **Disponibilidad Real vs. Objetivo:** Medición continua de tiempo en que el servicio está operativo y accesible.
- **Cumplimiento de Tiempos de Resolución:** Porcentaje de incidentes y solicitudes cerrados dentro de los objetivos pactados.
- **Nivel de Satisfacción del Cliente:** Índice promedio de las encuestas post-servicio.
- **Capacidad de Autoescalado Efectivo:** Número de veces que la plataforma respondió adecuadamente a picos sin intervención manual.

Estas métricas se presentan en dashboards accesibles a la dirección y se revisan periódicamente en comités de servicio.

Revisión y Mejora de Niveles de Servicio

- **Informes Periódicos:** Generación mensual y trimestral de reportes de SLA que incluyen tendencias, incumplimientos y causas raíz.
- **Reuniones de Revisión de Servicio:** Convocadas con stakeholders clave (negocio, TI, proveedores) para evaluar desempeño y negociar ajustes de SLA.
- **Acciones Correctivas:** Planes de mejora cuando se identifican desviaciones (optimización de arquitectura, refuerzo de equipos, ajustes de procesos).

Integración con la Cadena de Valor

- **Planear:** Determina los niveles de control y los requisitos de medición.
- **Involucrar:** Facilita la comunicación transversal de expectativas y resultados.
- **Entregar y Asistir:** Asegura el monitoreo activo durante la operación.
- **Mejorar:** Alimenta la práctica de Mejora Continua con datos objetivos de desempeño.

Prácticas técnicas relevantes

Además de las prácticas de gestión descritas, Storix se apoya en varias prácticas técnicas que proporcionan la capacidad operativa y la fiabilidad necesarias para

una plataforma basada en microservicios. Estas prácticas técnicas integran métodos y herramientas especializadas para asegurar que el diseño, despliegue y operación de los componentes de Storix cumplan los requisitos de disponibilidad, escalabilidad y seguridad.

Gestión de Implementación (Deployment Management)

Esta práctica coordina la liberación y el despliegue de nuevos componentes y versiones de microservicios en los distintos entornos (desarrollo, prueba, producción). Incluye:

- **Pipelines CI/CD:** Automatización de compilación, pruebas y despliegue utilizando herramientas como Jenkins, GitLab CI o GitHub Actions.
- **Entornos de Staging y Producción:** Definición de entornos idénticos que permiten validar cambios antes de su liberación y aplicar estrategias de “canary” o despliegues “blue/green” para minimizar riesgos.
- **Rollback Automatizado:** Mecanismos preconfigurados que revierten automáticamente a la versión anterior si se detectan errores críticos tras el despliegue.

Administración de Infraestructura y Plataforma

Se encarga de la provisión, configuración y monitorización de la infraestructura subyacente que ejecuta los contenedores de Storix y sus servicios de soporte:

- **Orquestación de Contenedores:** Uso de Kubernetes para gestionar réplicas, autoescalado, self-healing y distribución de carga entre nodos.
- **Infraestructura como Código (IaC):** Definición declarativa de recursos (VMs, redes, almacenamiento) mediante Terraform o Ansible, garantizando reproducibilidad y trazabilidad de cambios.
- **Monitorización y Alertas:** Implementación de Prometheus para la recolección de métricas de CPU, memoria, latencia y Grafana para la visualización en dashboards, junto con Alertmanager para escalado de incidencias.

Desarrollo y Gestión de Software

Garantiza que el ciclo de vida del desarrollo de cada microservicio sea ágil, consistente y de alta calidad:

- **Control de Versiones y Ramificación:** Uso de Git con estrategias de branching (feature branches, trunk-based development) para coordinar el trabajo en equipo y facilitar integraciones tempranas.
- **Pruebas Automatizadas:** Inclusión de pruebas unitarias, de integración y de contrato (contract testing) para validar que cada componente respeta sus interfaces y no rompe dependencias.

- **Revisión de Código y Seguridad:** Implementación de pull requests con revisiones por pares y herramientas de análisis estático (SonarQube, Snyk) que detectan vulnerabilidades y ayuda a mantener la calidad del código.

Integración con la Cadena de Valor

- **Obtener/Construir:** IaC y pipelines CI/CD agilizan la construcción y validación de componentes.
- **Diseñar/Transición:** Las estrategias de despliegue seguro (canary, blue/green) facilitan la transición de nuevas versiones.
- **Entregar y Asistir:** Kubernetes y monitorización continua garantizan la entrega estable y el soporte proactivo ante problemas.
- **Mejorar:** La retroalimentación de pruebas y métricas de despliegue impulsa ajustes en el pipeline y en la infraestructura para optimizar tiempos y reducir riesgos.

Desarrollo continuo

Requisitos Previos

- Jenkins instalado y funcionando (http://localhost:8080 o en un servidor).
- Plugin Git instalado en Jenkins.
- Repositorio Git local y remoto
- Acceso al servidor Jenkins desde el repositorio

Crear el Proyecto en Jenkins

1. Ir a Jenkins → *New Item* → elegir **Freestyle project**.
2. Asignar un nombre (por ejemplo: MiProyectoCI).
3. En la sección **Source Code Management**:
   - Elegir **Git**.
   - Poner la URL del repositorio.
   - Si es privado, agregar las credenciales.
4. En la sección **Build Triggers**:
   - Activar **Build when a change is pushed to GitHub** (si usas Webhooks).
   - O bien **Poll SCM** (para checar periódicamente si hay cambios)

COMANDOS A UTILIZAR

En la sección **Build**:

- Agregar un paso para compilar, por ejemplo:

  make build

Configurar Git Hook (si Jenkins y Git están en la misma máquina)

Dentro del repositorio local de Git, se puede usar el hook post-commit o post-push para que llame a Jenkins.

<tu_repo_local>/.git/hooks/post-commit

Contenido del hook (post-commit):

#!/bin/bash

curl -X POST http://localhost:8080/job/MiProyectoCI/build

Para permisos de autenticacion

chmod +x .git/hooks/post-commit

Configurar Webhook en GitHub/GitLab (alternativa más común)

1. Entra a tu repositorio en GitHub.
2. Ve a **Settings** → **Webhooks** → *Add webhook*.
3. En **Payload URL** pon:

   http://<IP_o_dominio_de_Jenkins>:8080/github-webhook/

1. **Content type: application/json.**
2. **Which events: elige Just the push event.**
3. Guardar.

OJO!!

Jenkins debe ser accesible desde Internet o estar en la misma red.

Token de Jenkins (si es necesario)

En Jenkins:

1. Clic en tu nombre de usuario → *Configure*.
2. En "API Token", generar uno nuevo si no hay.
3. Usar ese token en curl o al configurar el webhook si requiere autenticación.

Verificar Funcionamiento

1. Haz un cambio en tu código.
2. Realiza el push:

BASH

```
git add .

git commit -m "Cambio de prueba"

git push origin main
```

Comandos relevantes

```
# Inicializar repositorio Git

git init


# Configurar remoto

git remote add origin https://github.com/usuario/repositorio.git


# Añadir y hacer commit

git add .

git commit -m "Primer commit"


# Subir cambios

git push -u origin main
```

Usar webhooks es preferible a hooks locales si estás trabajando en equipo.

Si Jenkins no está en red pública, puedes usar herramientas como ngrok para exponerlo temporalmente.

Asegúrate de tener habilitado el puerto 8080 y de que Jenkins permita peticiones externas.

Entrega continua

Asegurarnos de:

- Jenkins instalado.
- Docker instalado y ejecutándose en la máquina de Jenkins.
- Plugin Docker Pipeline en Jenkins.
- Git (local o remoto) con la app de prueba.

APP.PY (ejemplo)

```
from flask import Flask

app = Flask(__name__)


@app.route("/")

def home():

return "Entrega Continua con Jenkins y Docker funcionando "


if __name__ == "__main__":

app.run(host="0.0.0.0", port=5000)
```

Documento requirements.txt

Flask

Dockerfile

FROM python:3.10-slim

WORKDIR /app

COPY . .

RUN pip install -r requirements.txt

EXPOSE 5000

CMD ["python", "app.py"]

Crear el Pipeline en Jenkins

1. Ve a Jenkins → *New Item* → **Pipeline** → MiAppCD.
2. En **Pipeline** → elegir opción *Pipeline script*.
3. Copia este ejemplo de pipeline declarativo:

pipeline {

agent any

stages {

stage('Clonar repositorio') {

steps {

git 'https://github.com/usuario/mi-aplicacion.git'

}

}

stage('Construir imagen Docker') {

steps {

script {

dockerImage = docker.build("mi-aplicacion:latest")

```
}
}
}
stage('Detener contenedor previo (si existe)') {
steps {
sh "docker stop miapp || true && docker rm miapp || true"
}
}
stage('Desplegar contenedor') {
steps {
sh "docker run -d -p 5000:5000 --name miapp mi-aplicacion:latest"
}
}
}
}
```

Configurar Jenkins para usar Docker

Verificar Docker en Jenkins

En Jenkins:

- Ve a: Manage Jenkins → Global Tool Configuration.
- Asegúrate que Jenkins tenga acceso a Docker.
- Si necesitas correr Jenkins con permisos de Docker:

COMANDOS

sudo usermod -aG docker jenkins

sudo systemctl restart Jenkins

Probar el Pipeline

1. Haz un cambio en tu repositorio.

2. Jenkins clonará el repo, creará la imagen, eliminará contenedores anteriores y levantará uno nuevo.
3. Visita: http://localhost:5000

Operación continua

Implementación de un Sistema de Monitoreo con Prometheus

Para asegurar la operación continua, es necesario contar con una herramienta de monitoreo que recoja métricas en tiempo real, alerte sobre anomalías y facilite la visualización gráfica del comportamiento de cada microservicio.

¿Por qué Prometheus?

Prometheus es un sistema de monitoreo y almacenamiento de series temporales diseñado para entornos de microservicios.

- Recopilación basada en “pull”, donde el servidor Prometheus interroga periódicamente a los microservicios para recopilar métricas expuestas en formato estándar.
- Lenguaje de consultas PromQL, que permite definir consultas y reglas complejas para calcular indicadores y disparar alertas.
- Alertmanager integrado, que gestiona notificaciones cuando las métricas superan umbrales definidos.
- Ecosistema amplio, con exportadores para Kubernetes, Docker, Linux, bases de datos, y bibliotecas clientes para instrumentar cada lenguaje de programación.

Monitoreo en Storix

La arquitectura de monitoreo con Prometheus se estructura en los siguientes componentes**:**

1. **Servidor Prometheus**
    - Ejecuta la recolección de métricas desde cada microservicio y desde componentes de infraestructura.

- Almacena series temporales en su base de datos interna de alto rendimiento.
- Evalúa las reglas de alerta definidas mediante PromQL y envía alertas a Alertmanager.

2. **Exportadores**
    - node_exporter: Recolecta métricas del sistema operativo de cada nodo (CPU, memoria, disco, red).
    - Advisor: Extrae métricas de contenedores Docker/Kubernetes, como uso de CPU y memoria por contenedor.
    - Instrumentación interna: Cada microservicio en Storix expone métricas personalizadas (número de solicitudes, latencia, errores) en un endpoint /metrics mediante bibliotecas oficiales de Prometheus.
3. **Alertmanager**
    - Recibe las notificaciones de alerta generadas por el servidor Prometheus tras evaluar las “alert rules”.
    - Gestiona la agrupación de alertas, los silenciamientos y reenvíos a canales externos.

Implementación de Prometheus

Para la implementación de Prometheus se utilizan servidores en Linux dedicados.

Primero se descarga la versión de Prometheus deseada, para buscar la versión se puede entrar a la página de descargas o al repositorio en Github para saber la versión más reciente. En este caso, se utiliza la versión 2.44

Descarga de Prometheus

wget
https://github.com/prometheus/prometheus/releases/download/v2.44.0/prometheus-2.44.0.linux-amd64.tar.gz

O con curl:

curl -LO
https://github.com/prometheus/prometheus/releases/download/v2.44.0/prometheus-2.44.0.linux-amd64.tar.gz

Después se instalan los binarios:

cd prometheus-2.44.0.linux-amd64

sudo mv prometheus promtool /usr/local/bin/

sudo mkdir -p /etc/prometheus /var/lib/prometheus

sudo mv console*.yml prom* /etc/prometheus/

sudo mv consoles/ console_libraries/ /etc/prometheus/

Se crea el usuario y se asignan los permisos:

sudo useradd --no-create-home --shell /bin/false prometheus

sudo chown -R prometheus:prometheus /usr/local/bin/prometheus /usr/local/bin/promtool

sudo chown -R prometheus:prometheus /etc/prometheus /var/lib/prometheus

Configurar y arrancar el servicio:

Se crea /etc/systemd/system/prometheus.service con el siguiente contenido:

[Unit]

Description=Prometheus Service

After=network.target

[Service]

User=prometheus

ExecStart=/usr/local/bin/prometheus \

  --config.file=/etc/prometheus/prometheus.yml \

  --storage.tsdb.path=/var/lib/prometheus

[Install]

WantedBy=multi-user.target

Se recarga y se corre el servicio de Prometheus:

sudo systemctl daemon-reload

sudo systemctl enable prometheus

sudo systemctl start prometheus

Configuración de servicios de Prometheus

Definir los “Jobs” que Prometheus debe sondear

En el archivo prometheus.yml se configura:

```
global:
  scrape_interval: 15s
  evaluation_interval: 30s

# Alerting
rule_files:
  - /etc/prometheus/alerts.yml

scrape_configs:
  # Prometheus se auto-monitorea
  - job_name: 'prometheus'
    static_configs:
      - targets: ['localhost:9090']

  # Monitoreo de nodos del clúster (node_exporter)
  - job_name: 'node_exporter'
    static_configs:
      - targets: ['node1:9100', 'node2:9100', 'node3:9100']

  # Microservicios de Storix (asumimos cada despliegue expone /metrics)
```

```
  - job_name: 'storix-services'
    kubernetes_sd_configs:
      - role: pod
    relabel_configs:
      - source_labels: [__meta_kubernetes_pod_label_app]
        regex: 'storix-.*'
        action: keep
      - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_scrape]
        regex: 'true'
        action: keep
      - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_port]
        target_label: __address__
        replacement: '$1:$/metrics'
      - source_labels:
[__meta_kubernetes_pod_annotation_prometheus_io_path]
        target_label: __metrics_path__
        replacement: '$1'
```

Definición de alertas con PromQL

En el archivo alerts.yml se definen las reglas:

```
groups:
  - name: storix_alerts
    rules:
      - alert: HighCPUUsage
        expr: avg(node_cpu_seconds_total{mode!="idle"}) by (instance) > 0.85
        for: 2m
        labels:
          severity: warning
        annotations:
          summary: "Uso de CPU alto en {{ $labels.instance }}"
```

```
      description: "El nodo {{ $labels.instance }} ha reportado un uso de CPU mayor al 85% durante 2 minutos."

  - alert: ServiceDown
    expr: up{job="storix-services", service=~"storix-.*"} == 0
    for: 1m
    labels:
      severity: critical
    annotations:
      summary: "Servicio abajo: {{ $labels.service }}"
      description: "El microservicio {{ $labels.service }} no responde desde hace más de 1 minuto."

  - alert: HighErrorRate
    expr: increase(storix_http_requests_total{code=~"5.."}[5m]) / increase(storix_http_requests_total[5m]) > 0.05
    for: 5m
    labels:
      severity: critical
    annotations:
      summary: "Alta tasa de errores (5xx) en storix-services"
      description: "Más del 5% de las solicitudes en los últimos 5 minutos han generado errores 5xx."
```

Configuración de Alertmanager

Prometheus gestiona la detección, pero Alertmanager se encarga de la notificación. En el archivo alertmanager.yml se configura:

```
global:
  resolve_timeout: 5m

route:
  receiver: "team-slack"
```

```
  group_wait: 30s
  group_interval: 5m
  repeat_interval: 3h

receivers:
  - name: "team-slack"
    slack_configs:
      - api_url:
'https://hooks.slack.com/services/T00000000/B00000000/XXXXXXXXXXXXXXX
XXXXXXXXXX'
        channel: '#alerts'
        send_resolved: true
  - name: "oncall-email"
    email_configs:
      - to: 'oncall@storix-company.com'
        from: 'prometheus@storix-company.com'
        send_resolved: true

inhibit_rules:
  - source_match:
      severity: "critical"
    target_match:
      severity: "warning"
    equal:
      - service
```

Y se despliega el Alertmanager agregando en prometheus.yml:

```
alerting:
  alertmanagers:
    - static_configs:
        - targets:
          - 'alertmanager.monitoring.svc.cluster.local:9093'
```

Ahora al acceder a http://IPPrometheus:9090, se puede acceder a la interfaz gráfica que brinda prometheus. En el enlace se sustituye “IPPrometheus” por la dirección IP del equipo o la máquina virtual que tenga el servicio de prometheus corriendo.

Con esta implementación de **Prometheus**, Storix contará con un sistema de monitoreo sólido y eficaz que no sólo detecta y notifica proactivamente incidentes, sino que también respalda la visibilidad continua del rendimiento de los microservicios y de la infraestructura subyacente.

Fases del ciclo de vida

El ciclo de vida del proyecto Storix, fundamentado en prácticas de Integración Continua y Despliegue Continuo (CI/CD), se organiza en las siguientes fases:

1. **Planificación y Definición de Requisitos**
   Antes de escribir una sola línea de código, se establecen los objetivos de negocio, los criterios de aceptación y los requisitos funcionales y no funcionales de cada microservicio. Se generan historias de usuario en el backlog, se establecen pipelines de CI/CD iniciales y se configuran los repositorios (control de versiones, ramas principales, flujos de trabajo de merge).

2. **Desarrollo de Funcionalidades**
   Cada desarrollador trabaja en su propia rama o “feature branch”, escribiendo código acompañado de pruebas unitarias totalmente automatizadas. La cultura de “commit temprano y a menudo” garantiza que cada cambio pequeño se valide rápidamente. Las pull requests

disparan pipelines de CI que ejecutan linting, compilación y tests, asegurando la calidad y coherencia del código.

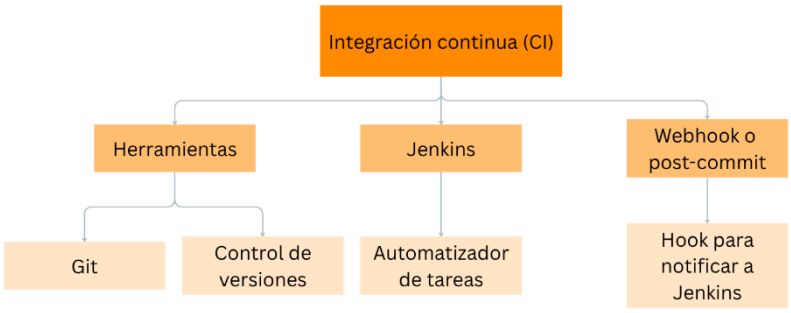
3. **Pruebas en Entorno Real**
   En producción, se combinan pruebas automatizadas de humo (smoke tests) con monitorización continua (Prometheus) para asegurar que los endpoints esenciales responden correctamente. Cualquier desviación dispara alertas al equipo de operaciones que, en caso necesario, detiene o revierte el despliegue.

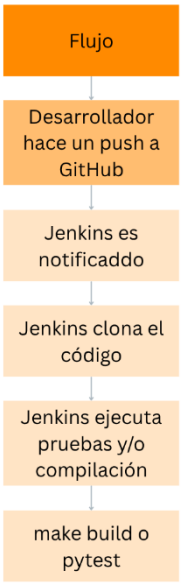
4. **Integración Continua**
   Cada modificación aceptada en la rama principal (main/trunk) dispara automáticamente un pipeline que ejecuta pruebas de integración: despliegue en un entorno de staging, ejecución de suites de tests de extremo a extremo, verificación de contratos entre microservicios y análisis de cobertura de código. Este paso valida que los componentes interactúan correctamente antes de avanzar.

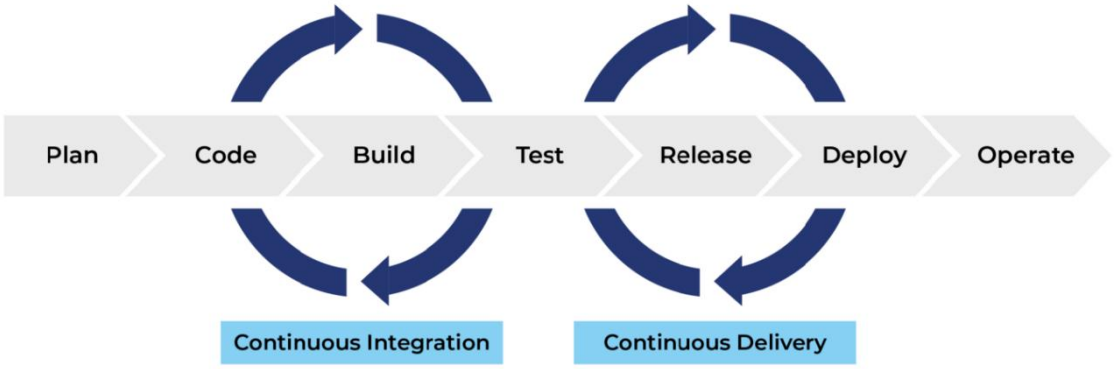
5. **Despliegue Automatizado**
   Una vez superada la fase de integración, se despliega la nueva versión en entornos de preproducción y producción siguiendo estrategias seguras. Los pipelines de CD orquestan la creación de artefactos, actualizan configuraciones de Kubernetes o de la infraestructura como código, y ejecutan migraciones de datos si fueran necesarias.

6. **Monitoreo y Retroalimentación**
   Tras cada despliegue se recogen métricas de rendimiento, logs estructurados y feedback de usuarios (encuestas de satisfacción). Estas evidencias alimentan dashboards de mejora continua, permitiendo identificar cuellos de botella, errores latentes o áreas de optimización.

7. **Mejora Continua y Ajustes**
   Con base en los datos operativos y el feedback, el equipo prioriza ajustes en el backlog: nuevas historias, refactorizaciones u optimizaciones de infraestructuras. El ciclo vuelve a iniciarse con la planificación de esas mejoras, cerrando así el bucle de CI/CD.







Conclusión

El proceso de desarrollo de Storix abarca desde la identificación de requerimientos hasta la operación y la mejora continua, garantizando que cada fase se ejecute con altos estándares de calidad y orientada a la generación de valor para la organización. La incorporación de los principios de ITILv4 en cada etapa asegura una gestión de servicios robusta y profesional, optimizando el rendimiento del sistema y proporcionando una base sólida para la toma de decisiones estratégicas. Con este enfoque, Storix se presenta como una solución innovadora y adaptable, preparada para satisfacer tanto las necesidades técnicas como los objetivos estratégicos en entornos empresariales dinámicos.