## Alcance del flujo

El flujo de CI/CD parte de un monorepo con las carpetas backend/ y frontend y del docker-compose que ya levanta ambos servicios. El objetivo es automatizar compilaciones y pruebas, empaquetar el backend y el frontend como contenedores versionados, publicarlos en Amazon ECR y desplegarlos en Amazon ECS Fargate detrás de un Application Load Balancer, con promoción controlada desde staging hacia producción. Todo se orquesta con AWS CodePipeline y se ejecuta en ejecutores efímeros de AWS CodeBuild, utilizando S3 como almacén de artefactos del pipeline, Systems Manager Parameter Store o Secrets Manager para parámetros y secretos, y CloudWatch para logs, métricas y alarmas.

## Preparación de infraestructura

Previo al primer release se define una VPC con subredes públicas para el ALB y subredes privadas para las tareas Fargate, junto con endpoints de VPC a ECR, S3 y CloudWatch para evitar NAT y reducir superficie de ataque. Se crean dos clústeres ECS, uno para staging y otro para producción, y en cada clúster se declara un servicio para el backend y otro para el frontend, cada uno con su definición de tarea, rol de ejecución y rol de tarea. El ALB expone HTTPS con certificados de ACM y enruta por host o por listener a dos target groups independientes, de modo que api.tu-dominio.com apunte al backend en su puerto 3001 y app.tu-dominio.com al frontend en su puerto 80. Se preparan los repositorios de ECR bluemed-backend y bluemed-frontend con escaneo de imágenes activado y políticas de ciclo de vida, y se cargan en Parameter Store o Secrets Manager las variables por entorno que consumirán las tareas en tiempo de ejecución. Finalmente, se habilita CloudWatch para logs estructurados y métricas, y se definen alarmas sobre latencia p95 y tasa de errores del ALB, así como sobre CPU y memoria de ECS, de forma que el pipeline y los operadores tengan señal objetiva.

## Orquestación y disparadores

La orquestación la realiza CodePipeline con una conexión CodeStar OIDC a GitHub para evitar credenciales estáticas. Un push a la rama develop inicia el flujo hacia staging y un push a main inicia la promoción hacia producción; los pull requests ejecutan compilaciones y pruebas sin desplegar. En la toma de código se captura el identificador de commit y se expone como variable de entorno a todo el pipeline para etiquetar imágenes, auditar cambios y facilitar rollback.

**Construcción y pruebas por componente**

La fase de construcción se ejecuta en paralelo para backend/ y frontend/ en proyectos separados de CodeBuild, cada uno con su directorio como contexto de trabajo. En ambos casos se instalan dependencias con npm ci, se ejecuta linting y la batería de pruebas unitarias y, sólo si todo pasa, se construye la imagen Docker usando los Dockerfiles que ya están en el monorepo. Cada imagen se etiqueta con latest y con la versión inmutable derivada del hash del commit; a continuación se publican ambas etiquetas en ECR. Este patrón proporciona un “last known good” práctico para evolución rápida y, al mismo tiempo, una etiqueta inmutable que preserva trazabilidad y permite revertir sin reconstruir. En esta misma fase se recomienda activar el escaneo de ECR y, si la política de la organización lo requiere, romper el build ante vulnerabilidades críticas para que los artefactos inseguros no progresen.

**Validación de integración con docker-compose**

Antes de tocar infraestructura, el pipeline ejecuta una validación de integración efímera reutilizando tu docker-compose. En un job de CodeBuild con Docker en modo privilegiado se levantan los dos servicios tal como los defines en el repositorio, se verifica la salud del backend golpeando /api/health en el puerto 3001 y se comprueba que el frontend sirva correctamente la raíz en el puerto 3000 y resuelva el fallback de SPA. Esta comprobación de humo, que dura minutos y se derriba al finalizar, captura incompatibilidades entre versiones de frontend y backend cuando ambas cambian en el mismo push y, al apoyarse en tu propio compose, asegura máxima fidelidad con el entorno de desarrollo.

**Despliegue automático a staging**

Superada la integración previa, CodePipeline aplica una acción de “Deploy to Amazon ECS” para cada servicio utilizando los imagedefinitions.json generados por los builds. La estrategia de actualización es rolling update, confiando en el ALB para marcar “healthy” únicamente las tareas que superan los health checks antes de drenar las anteriores. El backend se publica internamente en el puerto 3001 y el frontend en el puerto 80; las tareas inyectan sus variables de entorno desde Parameter Store o Secrets Manager mediante sus task roles, lo que evita secretos embebidos en imágenes o en scripts. Este diseño es suficiente para servicios stateless como los tuyos y minimiza la complejidad respecto a Blue/Green, manteniendo la disponibilidad durante el recambio.

**Pruebas post-deploy en staging**

Con los servicios ya corriendo en staging, el pipeline ejecuta pruebas de humo y, cuando aplica, pruebas end-to-end y de contrato contra las URLs reales. La SPA se valida con un recorrido básico que ejerce carga de los datos, navegación y estados vacíos, mientras que la API se verifica con colecciones que confirman códigos, payloads y encabezados CORS. Esta etapa es la principal puerta de calidad: sólo si staging se comporta correctamente se permite la promoción a producción. Si cualquiera de estas pruebas falla, el flujo se detiene y el equipo corrige antes de volver a intentar.

**Aprobación informada**

Antes de la promoción, el pipeline se detiene en un punto de aprobación manual que presenta evidencia útil: métricas recientes del ALB, estado y eventos de las tareas ECS, logs de las últimas ejecuciones en CloudWatch y el detalle del commit que se pretende promover. El objetivo es confirmar, con datos a la vista, que staging cumplió su cometido y que la promoción no introduce riesgo evidente.

**Despliegue a producción y reversión**

La promoción a producción repite la mecánica de staging con rolling update y con los parámetros y secretos propios del entorno. Si el comportamiento posterior al despliegue degrada la salud del sistema, la reversión no requiere compilar; basta con indicar al servicio de ECS la imagen etiquetada con el commit anterior que ya está disponible en ECR. Este proceso reduce el tiempo de recuperación y se apoya en la disciplina de etiquetado por hash implementada en la fase de construcción.

**Observabilidad y operación continua**

Una vez en producción, la observabilidad continua se basa en CloudWatch para logs y métricas, con dashboards que resumen latencia p95, tasa de errores, uso de CPU y memoria y distribución de estados de salud por target en el ALB. Las alarmas se configuran para notificar y, si se desea, bloquear promociones cuando la degradación es evidente. La incorporación de canarios sintéticos que pingen la ruta de salud de la API y la portada de la SPA aporta vigilancia activa que no depende de tráfico real. Si se necesita trazado de peticiones entre servicios, se puede instrumentar el backend con el SDK de AWS X-Ray y desplegar el daemon como sidecar en la tarea,conviene activarlo cuando se va a usar de verdad, para obtener valor sin añadir ruido ni costo innecesario.

**Seguridad, red y cumplimiento**

La seguridad se define con  tareas  de Fargate en subredes privadas, un ALB como único punto de entrada público, security groups de mínimo privilegio, endpoints de VPC para tráfico interno hacia ECR, S3 y CloudWatch, cifrado con KMS en buckets de artefactos y grupos de logs y políticas de IAM ajustadas a ARNs concretos para que CodeBuild y los servicios sólo accedan a lo estrictamente necesario. Los secretos se leen en tiempo de ejecución a través de los task roles, evitando variables en texto plano en el pipeline o dentro de las imágenes. Esta postura reduce la superficie de ataque y limita el alcance de un posible fallo de configuración dentro del monorepo.

**Cumplimiento de requisitos y evolución:**

El flujo descrito automatiza pruebas y builds para ambos componentes, despliega los contenedores del frontend y del backend en ECS Fargate detrás de un ALB, emplea servicios nativos de AWS como CodePipeline, CodeBuild, ECR, ECS, S3, Systems Manager y CloudWatch y presenta una descripción clara, paso a paso, de todo el proceso de CI/CD. Si más adelante el proyecto busca optimizar latencia global y costo del frontend, el pipeline puede incorporar una variante de despliegue a S3 y CloudFront sin tocar el backend ni la estrategia de pruebas, pero mientras el requisito contractual sea “contenedores para ambos”, mantener el frontend en ECS asegura cumplimiento literal y una operación homogénea.