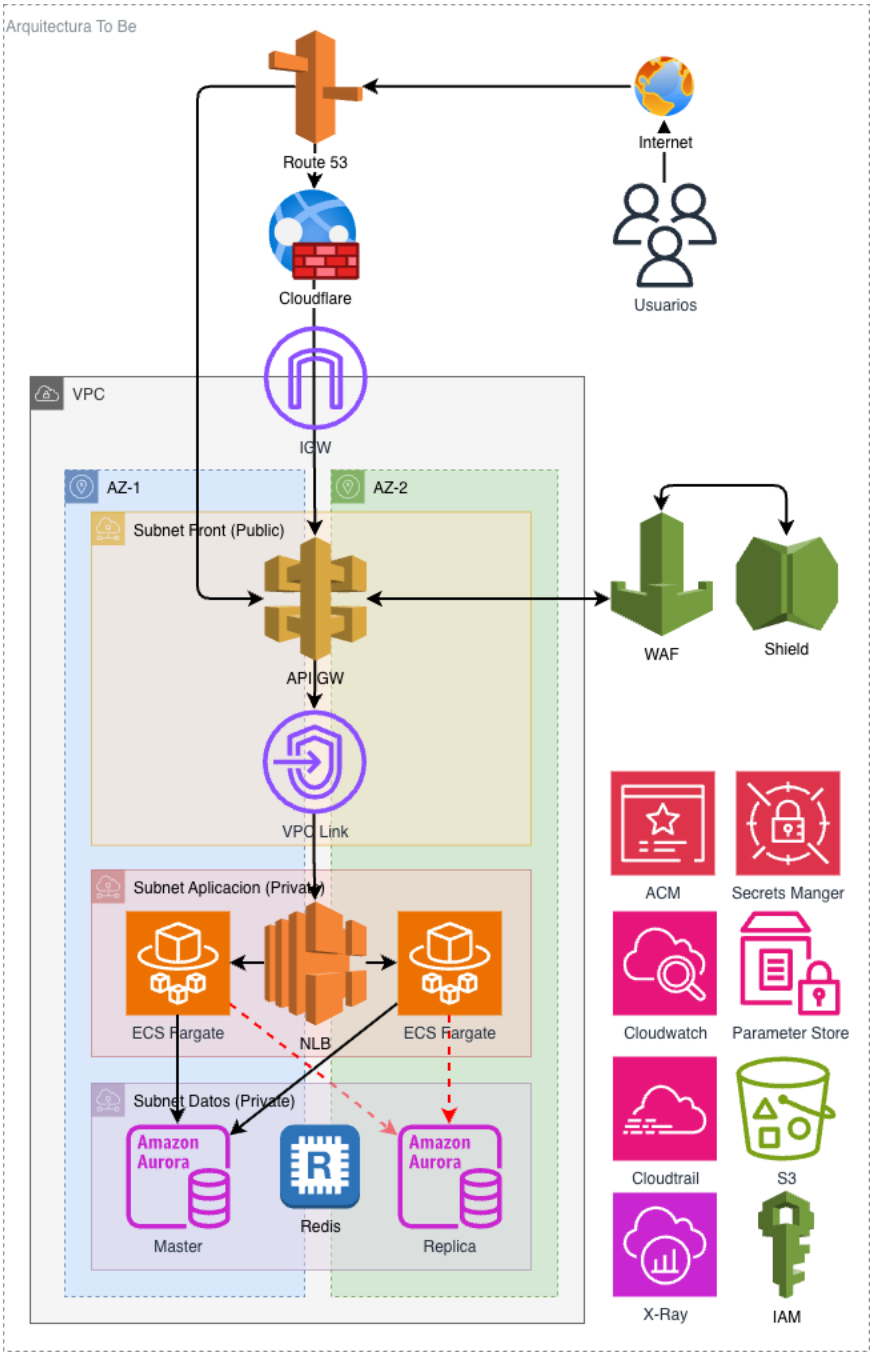
Propuesta Arquitectura de Seguridad Cloud



Propuesta de arquitectura TO-BE

1) Perímetro seguro y dual (Cloudflare / WAF+Shield)
Elimina exposición directa del API; dos caminos de entrada con inspección L7, mitigación DDoS y TLS gestionado (ACM).
Beneficio: *Aislamiento y defensa en profundidad con failover controlado por Route 53.*

2) API Gateway privado con VPC Link a NLB interno (L7→L4)
Centraliza *auth*, *rate-limit* y contratos; el NLB enruta tráfico TCP interno sin overhead HTTP.
Beneficio: *Reduce latencia y acoplamiento; elimina ALB públicos del AS-IS.*

3) Cómputo inmutable y multi-AZ con ECS Fargate
Tareas distribuidas por zona, autoscaling, circuit breaker y min capacity garantizan RTO ≤ 5 min.
Beneficio: *Sustituye instancias manuales; elimina SSH y parcheo de SO.*

4) Capa de datos resiliente: Aurora PostgreSQL + Redis Multi-AZ
Aurora con *writer/reader* replicados (RPO ≤ 5 min); Redis como *read-through cache* reduce lecturas.
Beneficio: *Consistencia fuerte en escritura y eventual en lectura; latencia < 20 ms.*

5) Red y servicios duplicados por AZ
Subredes, NAT Gateways, endpoints VPC, NLB targets y tasks ECS desplegados simétricamente.
Beneficio: *Tolerancia total a falla de una AZ sin pérdida de conectividad ni egress.*

6) Observabilidad 360° y gobierno IaC
CloudWatch/X-Ray/CloudTrail centralizados en S3 cifrado; Terraform + GitHub Actions (OIDC) controlan cambios y drift.
Beneficio: *Auditoría completa y reducción MTTR < 10 min.*

7) Optimización de latencia y costos operativos
Caching, API GW regional y NLB interno logran respuestas < 250 ms; Fargate y servicios managed reducen OPEX≈30%.
Beneficio: *Balance ideal entre rendimiento y costo.*

8) Trade-offs asumidos y mitigados conscientemente
Mayor costo (+25 %) por HA y complejidad gestionada con IaC y FinOps; consistencia eventual en Redis controlada con *write-around*.
Beneficio: *Arquitectura alineada con pilares AWS Well-Architected: Reliability & Operational Excellence.*

Terraform

1) Estructura modular y jerárquica por dominio
La IaC se organiza en módulos reutilizables que representan dominios lógicos:

- network/ (VPC, subnets, NAT, endpoints),
- security/ (IAM roles/policies, KMS, SG, WAF),
- compute/ (ECS, API Gateway, NLB, autoscaling),
- data/ (Aurora, Redis, S3),
- observability/ (CloudWatch, X-Ray, Trail, Config).
  Cada módulo tiene variables normalizadas (env, region, tags) y *outputs* exportables.

Beneficio: *permite actualizaciones por dominio sin afectar el resto; fomenta versionamiento semántico de módulos (v1.x, v2.x).*

2) Composición por entorno y cuenta
Cada entorno (dev, test, prod) se define en un *stack* separado bajo /live/<env> y se asocia a una cuenta de AWS distinta (Organizational Unit).
El *root module* de cada stack importa módulos de dominios y aplica configuraciones específicas (region, az_count, min_capacity, etc.).
Beneficio: *aislamiento total de fallos; facilidad para probar cambios por entorno; soporte nativo para multi-región.*

3) Gestión de estado remoto y bloqueo concurrente
El estado (terraform.tfstate) se almacena en **S3 cifrado con KMS**, y el *locking* se maneja con **DynamoDB**.
Cada entorno tiene su propio backend y tabla de *lock*.
Beneficio: *evita corrupción de estado; control de concurrencia y auditoría de cambios.*

4) Controles mínimos y buenas prácticas por defecto
Todo módulo debe cumplir:

- **Etiquetas obligatorias (CostCenter, Owner, Environment, DataClass, RTO, RPO)**.
- **Cifrado en reposo (KMS)** para S3, EBS, Aurora, Redis, Logs.
- **TLS 1.2+** para fronting (API Gateway / Cloudflare / NLB).
- **Bloqueo público total en S3** (block_public_access = true).
- **Política IAM mínima necesaria** por recurso.

Beneficio: *cumplimiento automático de seguridad base (CIS, NIST CSF, Well-Architected).*

5) Policy-as-Code (validaciones y cumplimiento automático)
Antes del plan, cada *pull request* ejecuta verificaciones automáticas:

- **tflint** (estilo y convenciones),
- **tfsec / Checkov** (seguridad y exposición pública),
- **OPA / Conftest** (cumplimiento de políticas corporativas: tags, cifrado, naming, versiones),
- **Infracost** (impacto financiero del cambio).

Beneficio: *los errores de seguridad o de costo se detectan antes del despliegue; IaC cumple con "shift-left security".*

6) Gestión de secretos y credenciales sin claves estáticas
No se usan aws_access_key ni secretos en texto plano.

- GitHub Actions se autentica con AWS mediante **OIDC federation** (sin credenciales permanentes).
- Variables sensibles provienen de **AWS Secrets Manager** o **Parameter Store**, no del repositorio.
- Los módulos consumen secretos vía data source y los exponen cifrados a Fargate.

Beneficio: *elimina riesgo de exposición de claves; rotación automática de secretos.*

7) Reutilización y portabilidad (parámetros vs defaults)
Los módulos definen variables clave (region, az_count, engine_version, retention_days, scaling_min/max) y mantienen *defaults seguros*:

- public_access = false
- encryption_enabled = true
- log_retention = 90d

Beneficio: *despliegues rápidos y homogéneos; sólo se sobrescriben parámetros necesarios.*

8) Versionamiento y control de drift

- Cada módulo se versiona en Git con *tags semánticos* (vX.Y.Z).

- Se programa un *workflow semanal* (terraform plan -detailed-exitcode) que detecta *drift* y abre un *issue* en GitHub si hay diferencias.

Beneficio: *evita configuraciones fuera de IaC; facilita auditorías y remediaciones automáticas.*

9) Entrega controlada y trazabilidad de cambios

Cada *PR* genera un terraform plan firmado, guardado como artefacto; los cambios a main/prod requieren revisión humana y *approval*.

Sólo se ejecuta terraform apply si el *planfile* aprobado coincide con el commit.

Beneficio: *seguridad operacional y trazabilidad completa por commit y autor.*

10) Documentación y auto-descubrimiento

- Cada módulo incluye README con entradas/salidas, ejemplos y dependencias.
- Uso de terraform-docs para generar documentación automática en el repositorio.

Beneficio: *facilita incorporación de nuevos miembros y reduce curva de aprendizaje.*

Mapa textual de carpetas (alto nivel)

```
/infra/
├── /modules/
│   ├── network/
│   ├── security/
│   ├── compute/
│   ├── data/
│   ├── observability/
│   └── iam/
├── /policies/
│   ├── opa/
│   ├── tfsec/
│   ├── checkov/
│   └── tflint/
├── /live/
│   ├── dev/
│   ├── test/
│   └── prod/
├── /workflows/
│   └── github-actions/
└── /docs/
    └── architecture/
```

GitHub Actions

1) **Flujo por Pull Request (PR)**

- Disparadores: push a ramas feature/bugfix y PR → main (o release/*).
- Puertas de calidad (shift-left): terraform fmt/validate, tflint, tfsec/Checkov, OPA/Conftest (tags, cifrado, S3 no-público, naming), Infracost (dif de costo).
- Terraform plan por *stack* (/live/dev|test|prod) con backend remoto (S3+DynDB lock).
- Publicación de artefactos: planfile firmado + reportes (lint, seguridad, costo) + sbom pequeño del módulo.

2) Flujo en rama protegida (main/prod) y ambientes

- Protecciones: revisiones requeridas, *status checks* obligatorios (todos verdes), “Require linear history”.
- Ambientes: dev (auto-apply), test (approval de plataforma), prod (manual approval + ventana de cambio).
- Apply solo descarga y ejecuta el planfile aprobado del PR mergeado (mismo checksum/commit).

3) Identidad sin secretos (permiso mínimo por job)

- GitHub → AWS con OIDC (federación), sin Access Keys.
- Roles separados por *job*: PlanRole (solo lectura, sts:AssumeRole limitado), ApplyRole (mutación restringida a recursos del stack), DriftRole (solo lectura).
- *Boundaries*: condiciones por sub (repo/ambiente), aud, branch, y *session duration* corta.

4) Artefactos, trazabilidad y evidencias

- Guardar planfile, reportes de seguridad/lint y *manifest* de cambios por 90 días.

- Anotar el PR con: costo incremental (Infracost), enlaces a dashboards, dif de políticas OPA y lista de recursos a crear/alterar/destruir.
- Registrar el State version post-apply y tag de módulos (vX.Y.Z) usados.

5**) Manejo de drift**
- Workflow programado (semanal o diario en prod) con terraform plan -detailed-exitcode.
- Si hay drift: crear issue con el *plan diff*, etiquetar al *owner* y (opcional) abrir PR de remediación automática en dev.
- Alertas a Slack/Teams con severidad por tipo de drift (seguridad/costo/funcional).

6) Rollback / mitigación
- Rollback inmediato = revert del commit que introdujo el cambio ⇒ nuevo plan ⇒ apply.
- Para incidentes: freeze de ambientes, *feature flags* en app, y change set limitado (solo destroy de lo recién creado si es seguro).
- Mantener "último planfile sano" como artefacto marcado para roll-forward controlado.

7) Orquestación por stack y concurrencia
- Matriz por /live/<env> (dev/test/prod) con concurrency groups por stack (evita *apply* simultáneos).
- Jobs network → security → data → compute → observability con dependencias explícitas (fail fast).

8) Cumplimiento y auditoría
- Bloquear apply si OPA/Conftest o tfsec fallan (no hay bypass en prod).
- Requerir *Change Request ID* en el PR para prod (anotación obligatoria).
- Exportar metadatos de cada *run* a S3 "audit-logs/ci" (JSON) y a un tablero de cambios.

Mini diagrama de secuencia textual (4 pasos)
1. Dev abre PR → Actions corre fmt/validate/lint/security/cost → terraform plan por stack → publica planfile y reportes como artefactos.
2. Revisión (arquitecto/owner) → todo verde → merge a main (rama protegida).
3. Workflow de main (ambiente): asume OIDC ApplyRole → descarga el planfile aprobado → terraform apply (con lock DynDB).
4. Post-apply: actualiza *state version*, publica evidencias, ejecuta *health checks* y notifica a Slack/Teams; programa verificación de drift.

Decisiones y trade-offs

Problema	Alternativas	Decisión	Riesgo residual / Mitigación
Base de datos para alta carga y consistencia	RDS estándar / Aurora PG Multi-AZ / DynamoDB	Aurora PostgreSQL Multi-AZ con *writer/reader* y PITR	Costo ↑ (~20%) y *replica lag* leve → mitigado con *query routing* y *monitoring*
Cómputo elástico sin servidores mutables	EC2 ASG / EKS / ECS Fargate	ECS Fargate (autoscaling + sin patching OS)	*Cold start* en picos → mitigado con *warm tasks* y *autoscaling thresholds*
Seguridad perimetral y entrada global	Cloudflare / WAF+Shield / ALB público	Modelo dual: Cloudflare (edge) o WAF+Shield (nativo AWS)	Doble gestión de reglas → mitigado con IaC unificado
Balanceo interno con baja latencia	ALB interno (L7) / NLB (L4)	NLB (L4) + VPC Link desde API GW	Menor visibilidad L7 → mitigado con X-Ray y métricas CW
Reducción de carga en DB	Redis / DAX / Ninguno	Redis Multi-AZ (read-through)	Consistencia eventual → mitigado con TTL + jitter + write-around
Centralizar logs y optimizar costo	CloudWatch / OpenSearch / S3+Glacier	S3 centralizado + *Lifecycle* + CloudWatch alerts	Búsqueda lenta en frío → mitigado con export temporal a OpenSearch
Gestión segura de secretos	Parameter Store / Secrets Manager / Vault	AWS Secrets Manager con rotación 90d	Fallas en rotación → mitigado con *grace period* y *retries*
Identidad segura en CI/CD	Access Keys / Runners privados / OIDC	OIDC GitHub→AWS con roles por *job*	Configuración de trust errónea → mitigado con validación inicial
Gobernanza multi-entorno	Única cuenta / Multi-cuenta con OUs	AWS Organizations + Control Tower	Sobrecarga administrativa → mitigado con Terraform modular y roles delegados
Monitoreo resiliente por AZ	Solo CW / CW + X-Ray / 3rd Party APM	CW + X-Ray + Synthetics por servicio	Costo de logs ↑ → mitigado con retención y filtros