Terraform

"Utilizamos Terraform para definir toda la infraestructura en AWS como código. Cada ambiente —dev, testing, staging y producción— estaba aislado usando **workspaces** o carpetas separadas con variables distintas. Las configuraciones incluían VPCs, subnets, grupos de seguridad, EC2, RDS y roles IAM. Cuando necesitábamos escalar o cambiar configuraciones, simplemente modificábamos los .tf y aplicábamos con terraform plan seguido de apply después de revisión.  
También usamos **remote backend** con S3 y bloqueo en DynamoDB para evitar conflictos entre equipos."

Docket y Kubernetes

"Aún no he trabajado directamente en la creación de pipelines CI/CD con GitHub Actions o Jenkins, pero tengo experiencia utilizando Docker en proyectos con TypeScript. Contenericé la aplicación, definí el Dockerfile y gestioné los volúmenes. Inicialmente tenía la base de datos dentro del contenedor, pero decidí moverla fuera para facilitar persistencia y administración.  
Aunque todavía no he trabajado con Kubernetes en producción, he estudiado sus conceptos clave —pods, deployments, services— y estoy familiarizado con su arquitectura. Es una tecnología que planeo dominar completamente en el corto plazo."

Prometheus, Grafana o ELK Stack

"En mi experiencia, usamos Splunk para el monitoreo de nuestras aplicaciones. Configuramos dashboards personalizados para visualizar logs en tiempo real, y detectar errores o validar que los procesos estuvieran activos correctamente. También usábamos alertas configuradas para eventos críticos.  
Aunque no he implementado Prometheus o ELK Stack directamente, conozco sus conceptos: Prometheus para recopilar métricas, Grafana para visualizarlas, y ELK para análisis centralizado de logs. Entiendo que son herramientas comunes en entornos Kubernetes y estoy listo para aprenderlas e implementarlas rápidamente si el equipo las utiliza."

BD relacionadas y no Relacionadas

| **Característica** | **Relacional (SQL)** | **No Relacional (NoSQL)** |
| --- | --- | --- |
| **Modelo** | Tablas (filas y columnas) | Documentos, clave-valor, grafos, columnas |
| **Esquema** | Fijo y estructurado (schema-first) | Flexible (schema-less o schema-late) |
| **Lenguaje** | SQL (SELECT, JOIN, etc.) | JSON, consultas específicas (MongoDB, etc.) |
| **Escalabilidad** | Vertical (más recursos en un solo servidor) | Horizontal (más nodos/distribuido) |
| **Ejemplos** | PostgreSQL, MySQL, Oracle | MongoDB, Cassandra, Redis, DynamoDB |
| **Uso típico** | Transacciones complejas, integridad | Big data, alta disponibilidad, agilidad |

**C++17 (y superior)**

* Mucho más expresivo, eficiente y seguro.
* Añade mejoras clave:
  + ✅ if constexpr
  + ✅ std::optional, std::variant, std::any
  + ✅ structured bindings (auto [x, y] = ...)
  + ✅ inline variables, constexpr if, mejoras a constexpr
  + ✅ std::filesystem (manejo de archivos multiplataforma)
  + ✅ Mejoras en std::thread, std::mutex, y sincronización
  + ✅ Menor boilerplate, más legibilidad
  + ✅ Más rendimiento (por mejores optimizaciones del compilador)

"Tengo mucha experiencia con C++ en entornos como Visual Studio 2013, por lo que conozco bien los fundamentos del lenguaje, incluyendo punteros, memoria dinámica, plantillas, y manejo de errores tradicional.  
Sin embargo, estoy también familiarizado con las mejoras que ofrece C++17, como std::optional, if constexpr, y structured bindings, que permiten escribir código más limpio y seguro. Estoy cómodo migrando o adaptando código legacy a versiones modernas, y me interesa seguir trabajando con estándares más recientes del lenguaje."