Daniel Saavedra

Estrategia de desarrollo del proyecto.

Estimación de esfuerzos

Estimar siempre será un acto de dar una visión de un *'más o menos tanto*' aún existen personas que les gusta oír el proyecto tomará en desarrollarse 3 meses, con 2 días y 4 horas. Siendo esto una mentira 100% que a los equipos de software, producto y manager les gusta creer.
En un mundo ideal donde exista confianza y trabajo de calidad el cliente pagará por los equipos el tiempo que se tarden en desarrollar el proyecto hasta el final, sin embargo, estamos aún un poco lejos a nivel de contratos ágil que permitan trabajar de esta manera junto a que pocos clientes aceptarán un contrato en el que no tengan unas fechas sobre las cuales proyectarse. Adicional a esto, trabajar sin fechas nos puede hacer caer en el síndrome del estudiante o cumplir la Ley de Parkinson.
Para afrontar lo anterior planteo de esta forma:

1. El equipo realiza un Inception en el que todos los involucrados del proyecto entiendan y listen todas las funcionalidades que debería tener el producto, es claro que ya se tiene una idea general y unas especificaciones, pero poder tener un espacio del equipo con el cliente es muy valioso y genera engagement.
2. Realizar una MDF (Matriz de descomposición funcional) donde se escriban cada una de las historias de usuario por más pequeñas que algunas puedan llegar a parecer y agruparlas por épicas.
3. A los dos equipos se les entrega la MDF para que cada uno realice la estimación de la siguiente forma:
    a. Equipo A: Trabaja con la técnica de Delphi donde cada miembro realiza una estimación en tiempo de cuánto tomarían las historias desglosadas y luego se hace un consenso por cada una de las historias.
    b. Equipo B: Estimación de tallas de camisas, se da un peso en tiempo a cada camisa XS, S, M, L, XL, XLLL donde realicen la votación con Scrum Poker.
    c. Con las dos votaciones hechas por los equipos se hace una comparación, en caso de estar cerca se toma un promedio y si están muy distantes cada equipo hace su apreciación de la votación.
    d. Con el tiempo final determinado se entrega al cliente el tiempo + un 20% de incertidumbre y solución de impedimentos que pueden llegar a ocurrir.

La Metodología

Para cualquier desarrollo de software existe incertidumbre, sin embargo, en este tipo observo que tiene una alta incertidumbre (manejo de dashboard, mecanismos de retoma, pagos) por lo que trabajar bajo una filosofía ágil inspeccionando y adaptando parece la mejor opción para hacer frente a dicha incertidumbre.
Los marcos para aplicar dicha metodología ágil puede variar dependiendo de la inspección que se realice sprint tras sprint, no obstante planteó el siguiente 'mix':

Equipos de máximo 7 personas, donde existan 2 junior, 2 semi senior, 2 senior y un PO (product owner) y por cada dos equipos tener un scrum master que ayude a gestionar las ceremonias y remover impedimentos. Trabajando con un marco principalmente *scrumban* tomando algunas cosas de otros marcos como XP para los pair programming, continuous integration, refactoring y TDD.
De esta forma no llamamos a nuestros equipos, como equipos scrum, sino en su lugar, llamamos un proyecto Agile, que toma lo mejor de cada marco de trabajo de acuerdo a la necesidad que se tenga.
Intencionalmente no agregue el rol del QC en el equipo debido a que la calidad es una responsabilidad de los Dev. Sin embargo, se puede considerar dicho trabajo que sea realizado por el product owner sólo como una verificación final a nivel producto previo a ser desplegado en producción

Los ambientes

Se contará con 3 ambientes:

Local

Donde cada uno de los dev contará con su ambiente que debe incluir su DB (docker) de tal forma que no colisione sus desarrollos con los de otro dev.

Dev (Serverless)

Este ambiente corresponderá a la rama Develop de gitflow, donde se integrarán los features una vez finalizados en sus respectivas ramas.

Test / UAT (Serverless)

Se desplegará las versiones pre productivas obtenidas desde la rama release de gitflow que consolida los features que estarán en producción. Se utilizará para una revisión final pre despliegue productivo.

Producción

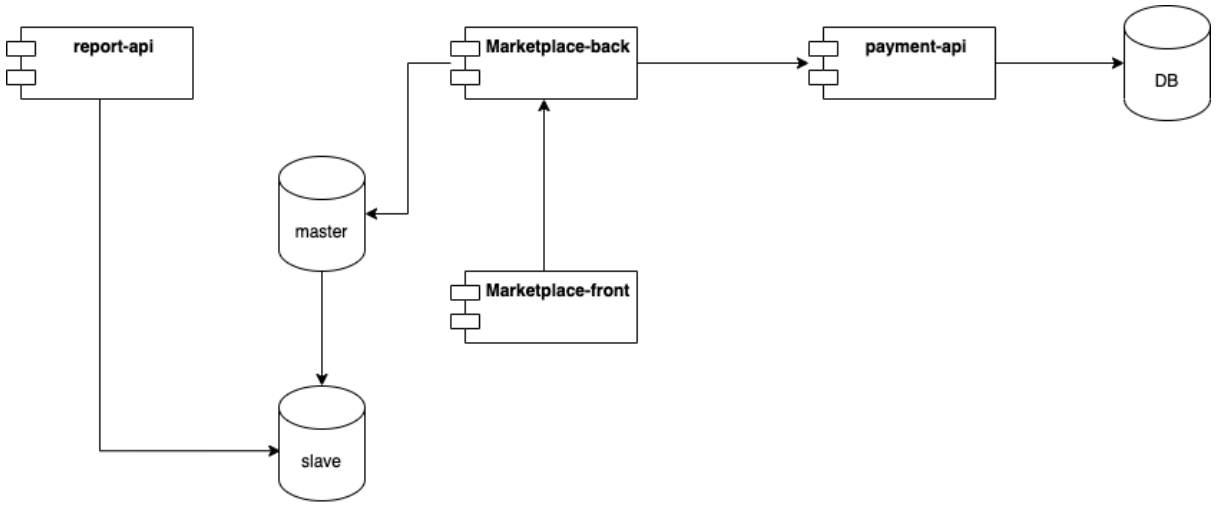
Ambiente del usuario final.

Cada PR (pull request) que se realice entre feature/develop/release/master se debe ejecutar la respectiva integración continua y para los ambientes no productivas se tendrá despliegue continuo, en los ambientes productivo se dejará hasta entrega continua.

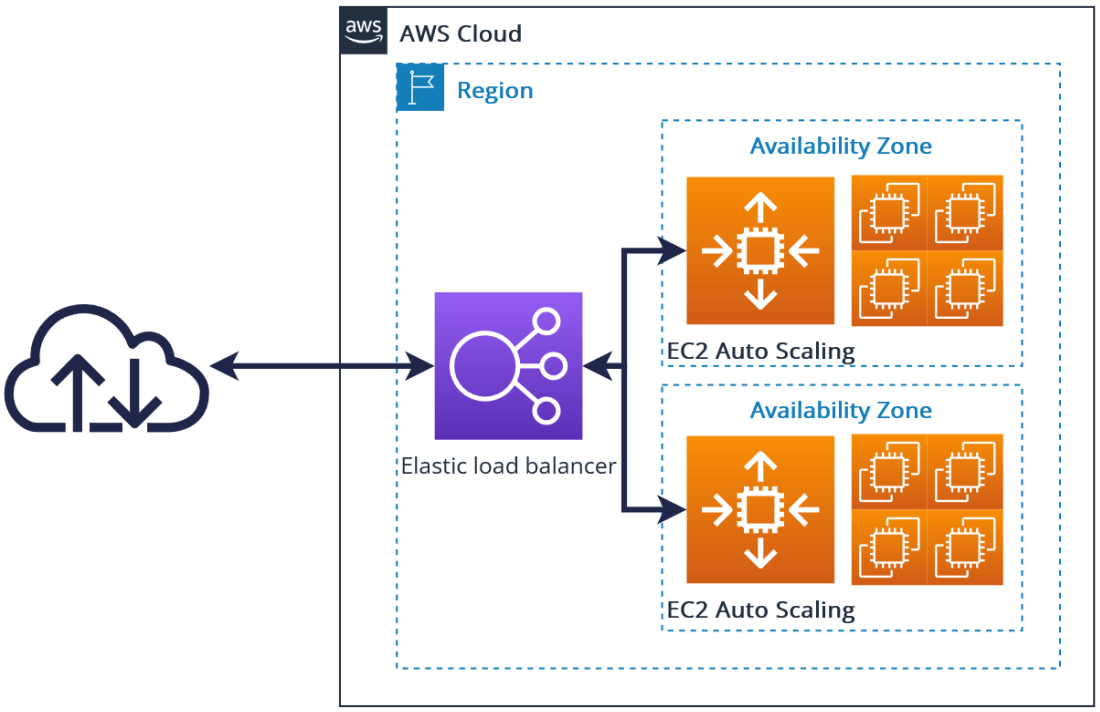
Daniel Saavedra

Propuesta Técnica

Vista general de componentes



Esta solución al recibir picos abruptos de transacción debe contar con una infraestructura elástica, dicha elasticidad se puede encontrar en aws. Donde podremos tener dos modelos, uno a través de ECS donde administremos las *task* que deben estar corriendo u otro en el que administremos la parte infraestructura como se presenta en el siguiente diagrama:



Daniel Saavedra

Escenarios de calidad

Los principales atributos de calidad (requerimientos no funcionales) con los que debe contar el aplicativo son los siguientes:

Availability: El sistema es disponible y accesible para los usuarios en la medida de lo posible teniendo en cuenta que es imposible lograr el 100%. Esto implica la implementación de redundancia en los servidores y la infraestructura, así como la detección y recuperación automática de fallos.
Para tener una correcta disponibilidad se deben aplicar las siguientes tácticas:

1. **Detecciones de fallas**:
    a. **Monitor**: Monitorear el estado de las aplicaciones del sistema junto con la infraestructura como puede ser memoria, CPU. Herramientas: Newrelic, Datadog
    b. Ping/Echo: Se intercambia un par de mensajes de solicitud/respuesta asincrónicos entre nodos; se utiliza para determinar la accesibilidad y el retraso de ida y vuelta a través de la vida. También se realiza durante el momento de deploy para verificar que quedó correctamente.
2. **Recuperación de errores:**
    a. Rollback: Durante la estrategia Devops implementada se debe también garantizar lograr hacer un rollback transparente

Medición:

1. **Source:** Grupo de servidores
2. **Stimulus:** Falla un servidor
3. **Artefact:** Servidor
4. **Environment:** Operación normal
5. **Response:** Sistema informa por email la anomalía y el sistema continúa la operación
6. **Response Measure:** No tiempo de indisponibilidad por falla de servidor

Performance: La solución debe ser capaz de manejar las peticiones a nivel backend en un umbral menor a 300 ms que brinde al usuario una grata experiencia de uso.
Para lograr un correcto performance se debe tener en cuenta el control de la demanda de recursos y el manejo de dicho recursos. Ej: si vamos a vender boletas de Taylor Swift previo a la habilitación de la boletería se debe calcular cuanto en X se debe incrementar dicha infraestructura previamente.

Medición:

7. **Source:** 10.000 usuarios
8. **Stimulus:** Iniciar con 400.000 request en un intervalo de 30 segundos
9. **Artefact:** Sistema
10. **Environment:** Operación normal
11. **Response:** Procesa todas las peticiones
12. **Response Measure:** Latencia media de 300 ms.

Security: Al manejar temas relacionados con pagos se debe garantizar el cumplimiento PCI DSS para el manejo de pagos, logramos seguridad aplicando las siguientes tácticas en nuestro aplicativo.

1. **Detectar ataques:**
    a. Detectar Intrusos con monitores
    b. Detectar ataques de denegación de servicios para bloquearlos
2. **Resistir ataques:**
    a. Autenticación de usuarios
    b. Autorización de usuarios
    c. Limitar los accesos
    d. Encriptar la data en tránsito y en reposo
3. **Recuperarse después de un ataque:**
    a. Buen manejo de logs para auditoría
    b. Manejo del no Repudio

Scalability: La solución debe ser escalable, lo que significa que puede seguir creciendo tanto en número de usuarios, los datos o productos que procesa y las solicitudes que recibe. Teniendo monitores que detecten el estado de la memoria y la CPU para aumentar o decrecer la infraestructura.
Para lograr lo anterior, se debe contar con todos los proyectos contenerizados, lo cual permite ejecutar un proceso DevOps más transparente.

Medición:

13. **Source:** 10.000 usuarios
14. **Stimulus:** Crece el número de usuarios en 10x en menos de 1 minuto
15. **Artefact:** Sistema
16. **Environment:** Operación horario pico
17. **Response:** Incrementa el número de servidores balanceando la carga correctamente
18. **Response Measure:** Latencia media de 300 ms, instalación de nuevas maquinas menor a 1 minuto.

Modifiability: La capacidad para agregar nuevas funcionalidades es algo prácticamente innato a todas las aplicaciones de software

Usabilidad: La interfaz de usuario es intuitiva y fácil de usar, lo que permite a los usuarios realizar operaciones de manera eficiente y sin confusiones.

Riesgos técnicos

- **Falta de documentación adecuada**: Muchos proyectos de software sufren la consecuencia de la materialización de este riesgo, la falta de documentación técnica y de usuario puede dificultar la comprensión y el mantenimiento del software. La documentación clara y actualizada es esencial para facilitar el mantenimiento, el soporte y la transferencia de conocimientos a otros miembros del equipo o usuarios. Una manera fácil de mitigar esto es sprint tras sprint revisar si se debe actualizar la

deuda técnica. Adicional, mantener la documentación a su mínima expresión posible documentar de más siempre será perjudicial.

- **Falta de experticia técnica del equipo**: Si el equipo de desarrollo no cuenta con las habilidades y experiencia adecuadas para enfrentar los desafíos técnicos del proyecto, puede haber retrasos en el desarrollo, baja calidad del software y falta de capacidad para solucionar problemas técnicos complejos.

- **Gestión inadecuada de la configuración**: Si no se gestiona correctamente la configuración del software, como las dependencias externas, las variables de entorno o las configuraciones del servidor, pueden ocurrir problemas de despliegue y configuración incorrecta en diferentes entornos.

- **Problemas de calidad del código**: Si el código fuente no cumple con los estándares de calidad, puede volverse difícil de mantener, modificar y extender en el futuro. La falta de pruebas unitarias adecuadas y la falta de estructura y modularidad pueden afectar negativamente la calidad del código.
  Este riesgo se puede materializar si no se hace conciencia a pleno con los equipos de desarrollo acerca de la importancia de la calidad.

- **Riesgos de cumplimiento legal**: Dado que el software maneja dinero, existen riesgos de incumplimiento de leyes y regulaciones de protección de datos. Es importante cumplir con las normativas vigentes y adoptar medidas de seguridad adecuadas para proteger la privacidad y la seguridad de los datos.

- **Riesgos de seguridad**: La seguridad del software es crucial para proteger la información y los sistemas. Las vulnerabilidades de seguridad, como los fallos de autenticación, la inyección de código malicioso o la falta de cifrado, pueden poner en riesgo los datos y la reputación de la empresa. Es importante realizar pruebas de seguridad y seguir buenas prácticas de desarrollo seguro.

Gestión

Gestión del equipo desde el punto de vista técnico:

Definir roles y responsabilidades: Es importante establecer claramente los roles y las responsabilidades técnicas dentro del equipo, asegurándose de que cada miembro tenga una comprensión clara de lo que se espera de ellos.

Asignar tareas adecuadas: En el momento de realizar un sprint planning y el equipo esté dividiendo las tareas iniciales, velar por que las tareas sean asignadas de acuerdo a capacidades. En un comienzo de sprint no repartir todas las tareas sino sólo entregar una tarea a cada miembro, después cada uno al momento de ir finalizando su tarea va tomando más actividades.

Comunicación efectiva: Fomenta una comunicación clara y abierta dentro del equipo, promoviendo la colaboración y el intercambio de conocimientos técnicos.
Enviar el uso del email dentro de miembros del equipo y procurar el uso de videollamadas para resolver dudas o discusiones.

Fomentar el desarrollo profesional: Espacios de capacitación semanal con el equipo a nivel de arquitectura de software o temas que sean de interés para el equipo. Esto ayudará a mantener actualizadas sus habilidades y motivación.

Medición y control del proceso

Las siguientes serán métricas que se tomarán de cada proyecto sprint tras sprint y verificar que no se degrade y en caso de degradarse deberá mejorarse en sprint siguientes:

- Coverage: Porcentaje de pruebas unitarias (unit test con mutation test)
- Response Time backend reads: Tiempo de respuesta promedio de lectura del backend
- Response Time backed write: Tiempo de respuesta promedio de escritura del backend
- Error 5XX: Porcentaje de peticiones que quedan con error 5XX
- Error 4XX: Porcentaje de peticiones que quedan con error 4XX
- Cantidad de errores productivos
- Número de archivos en PR: Limitar y medir la cantidad de archivo de un PR permite que sea más fácil la revisión de los PR.

Manejo de la deuda técnica

La deuda técnica es algo que va de la mano de los productos de software, un proyecto de software puede no tener más deuda técnica siempre y cuando no se esté trabajando más sobre el. Si se está trabajando sobre el productor sprint tras sprint entonces se plantea los siguientes mecanismos para trabajarla y mitigarla.

1. Implementación de **mutation test** para garantizar calidad en las pruebas unitarias
2. Identificación de deuda técnica: herramientas como **sonarqueue** ayudan en este trabajo
3. Priorización: Cada sprint se debe reservar un % de tiempo para trabajar la deuda técnica.
4. Revisiones periódicas y aleatorias por parte del TL para buscar cosas por mejorar.
5. Integración continua.