Moviri è una società di consulenza informatica e una softwarehouse specializzata nelle aree di: **Performance engineering**, Analytics, Cybersecurity, Internet of things.

Nel seminario si tratterà solamente dell'area di performance engineering. Con specializzazione sulle performance non si intende esclusivamente sulla velocità di un applicativo ma anche riuscire a progettare e monitorare degli applicativi in grado di erogare dei servizi soddisfacenti nel tempo.

Le aree su cui si concentrano per le performance sono:

1- Design & validation: esecuzione di test di perfomance, tuning, implementazione di una pratica di continuos testing, settare dei quality gates da rispettare per determinati parametri di performance dell'applicativo, ecc…
2- Self-driving Ops: automatizzare, tramite l'utilizzo di prodotti che si basano sull'intelligenza artificiale, il monitoraggio dei sistemi e individuare dei comportamenti anomali ed eseguire le dovute correzioni sul sistema per risolvere eventuali problematiche
3- Observability: avere consapevolezza di come vengono erogati i servizi digitali andando a monitorare le performance, l'esperienza degli user, la IT performance e eseguire delle analisi sui dati ottenuti tramite anche la data visualization
4- Planning & control: modificare le risorse affidate ai vari settori sulla base dello sviluppo dell'applicativo o del suo mantenimento, concentrandosi in particolare sulla scalabilità del sistema in quanto il bacino d'utenza potrebbe crescere col tempo e con esso anche l'applicativo deve essere modificato affinché possa erogare i servizi ad un sempre più crescente quantitativo di utenti.

Ci sono molti **servizi che vengono forniti dagli specialisti di Moviri**, alcuni dei più frequenti sono:

- Peak Demand Events: classico esempio del Black Friday, ovvero momenti in cui vi sono dei picchi d'utenza che potrebbero compromettere le funzionalità dell' applicativo.
- Datacenter & Cloud Optimization: ottimizzazione di sistemi data center cloud o in loco con ottimizzazione dei costi e risorse
- IoT & Client Side Performance: gestire le performance per tipologia di device utilizzato come web, mobile, ecc..

Le performance sono molto importanti per diversi motivi, vediamo alcune conseguenze di una performance carente applicate ad un esempio di un applicativo di e-commerce:

- Bounce rate: se il caricamento di una pagina web che offre un servizio è lento, tendenzialmente gli utenti perderanno interesse nell'applicativo
- Negative Pubblicity: le scarse performance provocano una cattiva pubblicità da parte dei clienti insoddisfatti dei servizi offerti causando non solo la perdita di clienti insoddisfatti, ma anche un decremento dei nuovi clienti
- Brand Reputation: la lealtà nei confronti di un brand può essere minata dalle pessime prestazioni di un loro prodotto
- Forever Lost: il 23% degli utenti insoddisfatti che abbandonano il sito non lo riutilizzeranno mai
- Producitivty Loss: la produttività degli utenti di un applicativo (quindi non i clienti nel caso dell'ecommerce ma del backend) ne risente se le performance non sono ottimali

Tutte queste conseguenze ad una cattiva performance di un sistema hanno due conseguenze catastrofiche:

- Competitors Advantage: i competitors che offrono servizi analoghi avranno enormi vantaggi se il loro prodotto è più performante, causando una perdita di clienti ingente e conseguente guadagno
- Revenue Drop: il guadagno perso a causa dei problemi di efficienza nel mondo raggiunge i bilioni

Un esempio di come il tempo di caricamento di una pagina possa influire sulle prestazioni e conseguentemente i guadagni di un applicativo si può vedere come **Amazon** se aumentasse il tempo di caricamento delle pagine di 0.1 secondi perderebbe circa l'1% dei suoi guadagni. Questa stima potrebbe sembrare bassa ma rapportata ai miliardi guadagnati da Amazon anche l'1% è tanto. Inoltre si può dire che incrementare di così poco il tempo di caricamento di una pagina non sarebbe percettibile all'occhio umano, ma nonostante ciò si ha comunque un impatto sulle performance e i guadagni finali.

I clienti però spesso non fanno tutto ciò che è stato spiegato prima ma si devono affidare ad esterni. I motivi sono vari tra cui:

- Tools: non avere la disponibilità o la conoscenza dei tools per eseguire performance testing
- Skills: non sempre le compagnie hanno persone specializzate nel testing
- Time: il tempo richiesto per il testing è abbastanza consistente e spesso le aziende non hanno a disposizione tempo e risorse da allocare a questo step, soprattutto quando rispettare le deadline e il tempo di mercato costringe le aziende a rilasciare applicativi e aggiornamenti sempre più frequentemente
- Enviroments: le aziende faticano a simulare un ambiente di test uguale o verosimile all'ambiente di rilascio dell'applicativo

Ora andiamo a vedere come Moviri adotta gli **approcci** appena menzionati per andare ad eseguire le performance testing degli applicativi.

Il Sistema è performante rispetto a diverse metriche non solo la velocità. Il sistema in generale deve essere testato rispetto alle metriche che definiscono le performance.

La prima fase è di **assesment e design**. L'obiettivo di questa fase è quello di determinare il carico di lavoro e analizzare il contesto per lo sviluppo dell'app. Per fare ciò si deve:

- Ottenere informazioni di business interrogando chi commissiona il lavoro
- Definire obiettivi tecnici e target
- Analizzare le statistiche di produzione e\o la production forecast, ad esempio tenere conto del bacino d'utenza del prodotto alla release dello stesso per garantire il servizio tenendo anche conto del tipo di applicativo che si sta sviluppando (su cloud oppure no)
- Definire un business process e i vari scenari, ad esempio la navigazione su un sito di acquisti con tutti i click necessari per eseguire l'acquisto. In questo passaggio bisogna tenere conto che fare performance test è diverso da fare functional test. Per il primo bisogna concentrarsi sulle funzionalità critiche del sistema, per il secondo si può guardare più in generale il funzionamento del programma.
- Definire il test data-set. Un ambiente di test artificiale deve cercare di emulare al meglio il prodotto finale al fine di ottenere risultati utili. Bisogna prestare attenzione anche alla tipologia di utente che fa uso dell'applicativo e testare le funzionalità per ognuno di essi; in particolare bisogna non solo testare le funzionalità del cliente di un sito di acquisti ma anche di tutti quegli utenti che fanno attività di back office come sportellisti, ecc. Bisogna anche tenere conto del bacino d'utenza in quanto eseguire un testing su un utilizzo in parallelo dell'applicativo di pochi utenti quando in realtà il prodotto finale è destinato ad un ampio numero di utenti, genera dei risultati poco utili.

La seconda fase è la fase di **build**.

- Registrazione dello user path. Qui si inizia a cerare il test. In generale si registrano le chiamate al server da un proxy per poter creare in automatico un test.
- Migliorare lo script con dei check e correlazioni. In questo caso si va a parametrizzare il case test definito dalla registrazione delle azioni dell'utente. In questo modo utilizzando vari parametri si possono simulare diversi tipi di scenari con diversi input. Bisogna anche tenere contro della correlazione, ovvero il fatto che eseguire un azione (ad esempio di login) avrà delle ripercussioni sul funzionamento dell'applicativo come ad esempio avere un'interfaccia da cliente che acquista oppure da admin.
- Model the Load. Definire il modello di carico va valutato in base al prodotto commissionato. L'applicativo può avere determinate fasce orarie in cui il carico è molto grande e altre fasce in cui il carico è lieve.

Si possono fare diversi tipi di test per il carico di lavoro:

- Load test: vado a vedere come funziona il mio applicativo nello scenario di carico standard
- Stress test: test che mira ad identificare quando il sistema inizia a faticare o addirittura a produrre dei crash
- Endurance test: test che mira a verificare che l'applicativo in carico di lavoro normale ma su un lungo periodo di tempo funzioni a dovere
- Spike test: test che mira a vedere come reagisce il sistema ad uno spike di carico di lavoro
- Break point test: test che va a misurare quando il sistema risponde con delle metriche che sono fuori dalle soglie attese (ad esempio tempi di risposta troppo lunghi durante la fase di login

La terza fase è la fase di **execute**. Per eseguire il test Moviri usa dei tool enterprise.

- Production like infrasctructure\ architecture: simulare I dispositivi che andranno ad utilizzare il sistema (caratteristiche hardware,SO, ecc)
- Noisy negihbors: bisogna tenere conto che su un sistema vengano utilizzati altri applicativi che condividono le risorse hardware.
- External, third party services sizing\mocking: elementi di terze parti che non possono essere controllati vengono mockati per simularne le funzionalità
- Database sizing
- Load farm sizing and location: per un test che deve simulare l'utilizzo di diverse migliaia di utenti si fa uso di alcuni load injector che simulano le azioni di un grande numero di utenti virtuali. Bisogna anche tenere conto della location dell'applicativo che può essere ad esempio una intranet oppure degli endpoint api che vengono utilizzati da applicazioni mobile in tutto il mondo.

La quarta fase è la fase di **analyze**. Una volta eseguiti i test si analizzano i risultati nei seguenti modi:

- Black box: analisi che si concentra solo sulle metriche end user come validazione e non regression case. Il mio sistema è una scatola nera dove il sistema ha dei tempi di risposta ma non so se sono dei tempi di risposta adeguati. In questo caso si valuta ad esempio quanti utenti riesce a sostenere il sistema. In questo caso i fenomeni di collo di bottiglia non possono essere identificati.
- White box: analisi che mira a un full stack understanding del Sistema target. In questo caso possiamo andare a monitorare tutte le chiamate del sistema e avere evidenza degli errori delle chiamate e tracciare anche i tempi di risposta. In questo caso i fenomeni di collo di bottiglia possono essere identificati e si possono anche individuare possibili tuning per evitarli.

Una fase di mezzo è la fase di **tune**. Dopo avere eseguito l'analisi si avvia una fase di tuning per migliorare il sistema tramite una retro azione. Poi si ripartirà dalla esecuzione dei test finchè non ci si ritiene soddisfatti.

La quinta fase è la fase di **deploy**. In questa fase ci sono diversi tipi di documenti che si possono consegnare chiamati Deliverables. Si può mandare un delivery report dove si illustrano i test e come questi sono stati superati o meno. Si fa anche un final report che esegue un'analisi dettagliata del comportamento dell'applicativo rispetto ai risultati che ci si aspetta, rispetto ai problemi di collo di bottiglia.

Un concetto importante è il concetto di **quality gate**. Anche se un test viene superato bisogna verificare che il sistema rispetti delle metriche predefinite affinchè il test sia effettivamente superato. Una volta identificati i service level indicator si va ad identificare i service level object, ovvero le metriche che definiscono come un sistema dovrebbe lavorare. Il quality gate viene inserito alla fine del processo di testing. Questa fase determina se l'applicativo è rilasciabile oppure se bisogna eseguire un ulteriore fase di tuning.

È importante portare una parte di testing in **automatizzazione**, ovvero senza la necessità di un occhio umano. Questo perché bisogna automatizzare li script affinchè avvenga un continuo performance test con le varie release dell'applicativo. Ci sono due tipi di testing:

- E2E testing: testing che richiede un grande quantitativo di tempo che è avvezzo a fallimenti in casi di modifiche di sistema e che diventa poco utile con lo scalare dell'ambiente di test
- **Component continuos testing**: test veloce che può essere applicato ad un singolo servizio o ad un insieme limitato di essi che consente di seguire le tempistiche di mercato con continue release e feedback

Il fatto di avere dei quality gate automatici consente di avere dei performance test ripetibili.

Akamas è un applicativo sviluppato da Moviri che consente di andare a modificare le configurazioni del sistema andando a migliorare i risultati che si vogliono ottenere. Questo applicativo va a modificare i parametri di configurazione della VM in maniera automatica consentendo di eseguire diversi test automatizzati che valutano le performance senza la necessità di un tuning umano.

I livelli di **maturity model** che le aziende possono avere rispetto al testing sono:

- Fire fighting: richiesta di assistenza perché ci sono dei malfunzionamenti gravi. In generale non vengono fatte validazioni delle app ne tuning che portano a questi problemi
- Repeatable: i clienti capiscono che non possono trascurare le performance, definiscono dei quality gate e fanno alcuni tuning
- Managed: ho dei quality gate obbligatori. Il tuning è strutturato e viene eseguito per tempo.
- Optimized: livello di automation alto dove tutto viene testato in automatico compresi testing e tuning

Sulla base del livello di maturity del cliente, Moviri ha differenti approcci (**Delivery Models**)che variano di intensità. Nel caso di fire fighting in generale mandano una task force che si occupa di una azione immediata. Nel caso di repeatable i clienti richiedono un'assistenza a moviri che fornisce un long term plan di test di performance. Nel caso di managed un team dedicato di Moviri presidia assieme ai clienti le fasi di testing e tuning durante lo sviluppo. Nel caso optimized il team dedicato non si occupa più di performance test ma di tematiche di testing a 360 gradi.

MAPPA CONCETTUALE ARGOMENTI TRATTATI