VANTAGGI APP WEB MONOLITICHE:

* Più facili da sviluppare perché lo stack tecnologico è uguale a tutti i livelli (non cambia).
* Più facile da testare (test di integrazione e end-to-end) perché l’applicazione è tutta raggruppata.
* Più semplice e facile da deployare. C’è solo un elemento di cui occuparsi.
* Più semplice da scalare perché basta moltiplicare il numero di istanze tramite un bilanciatore del carico.
* Richiede team più piccoli per mantenere l’applicazione.
* I membri dei team condividono più o meno lo stesso insieme di skill.
* La tecnica a stack è semplice e il più delle volte si impara facilmente.
* Lo sviluppo iniziale è più veloce, quindi il tempo di commercializzazione è più rapido.
* Richiede infrastrutture semplici. Anche un semplice application container o JVM sarà sufficiente per avviare l’applicazione.

SVANTAGGI APP WEB MONOLITICHE:

* I componenti sono strettamente legati tra loro (bassa indipendenza) e questo può generare dei side effects su un certo componente quando se ne sta modificando un altro.
* Aumenta la complessità dell’applicazione con conseguente rallentamento dello sviluppo.
* L’intera applicazione ha bisogno di essere rideploiata per eventuali cambiamenti.
* L’applicazione è meno affidabile a causa dei moduli strettamente legati. Un piccolo problema in un servizio potrebbe rompere l’intera applicazione.
* L’adozione di nuove tecnologia è difficoltosa in quanto bisognerebbe migrare l’intera applicazione. Spesso la migrazione incrementale non è possibile e, quindi, molte applicazione monolitiche finiscono con l’avere uno stack tecnologico obsoleto.
* I servizi critici non possono essere ridimensionati singolarmente con il conseguente aumento dell’uso delle risorse poiché l’intera applicazione dovrà essere ridimensionata.
* Enormi applicazioni monolitiche avranno un tempo di avvio più elevato e un uso maggiore delle risorse in termini di CPU e memoria.
* I team saranno più interdipendenti e sarà più complicato scalare i team.

ARCHITETTURA A MICROSERVIZI

L’architettura a microservizi sta avendo molta popolarità negli ultimi anni anche grazie alla sua modularità e alla sua scalabilità. Tale architettura può offrire quasi tutti i vantaggi di quella monolitica con più caratteristiche e più flessibilità. A differenza dell’architettura monolitica, però, è abbastanza difficile generalizzare un architettura a microservizi in quanto potrebbe variare notevolmente nel tempo. Elementi che contraddistinguono i microservizi:

* I componenti dei microservizi sono meno accoppiati. Essi possono essere sviluppati, testati, deployati e scalati indipendentemente senza avere side effects sugli altri componenti.
* Ogni componente può avere il proprio stack tecnologico e utilizzare un differente linguaggio di programmazione.
* Spesso usano caratteristiche avanzate come il service discovery, circuit breaking, load balancing ecc
* I componenti di un microservizio sono molto più leggeri perché svolgono specifiche funzionalità.
* Spesso hanno un monitoraggio e una risoluzione dei problemi estesa.

VANTAGGI MICROSERVIZI:

* Componenti più indipendenti con un conseguente isolamento e un app più facile da testare e più veloce all’avvio.
* Modifiche e tempi di commercializzazione più rapidi.
* I servizi possono essere deployati in maniera indipendente facendo applicazioni più affidabili e più semplici da fixare.
* Eventuali problemi in un servizio non compromettono l’intera applicazione, ma solo quello specifico servizio.
* L’adozione di nuove tecnologie è semplice, i componenti possono essere aggiornati indipendentemente nella migrazione incrementale, rendendo possibile avere uno stack diverso per ciascun componente.
* Si possono stabilire dei modelli di scalatura più efficienti in modo da usare l’infrastruttura in maniera più efficiente.
* I singoli componenti si avvieranno più velocemente rendendo possibile parallelizzare e migliorare l’avviamento generale.
* I team saranno meno dipendenti gli uni dagli altri. Molto adatto per team che usano metodologie di lavoro agili (come la scrum agile).

SVANTAGGI MICROSERVIZI:

* Più complesso in termini di stack complessivo visto che i diversi componenti potrebbero avere stack tecnologici diversi fra loro e il team potrebbe essere rallentato da ciò.
* Può essere difficile eseguire test end-to-end o di integrazione in quanto vi sono diverse parti che cambiano dinamicamente nello stack.
* L’intera applicazione è più complessa da implementare perché sono presenti complessità con i container e la virtualizzazione.
* Scalare è più efficiente ma l’upscaling (miglioramenti di qualità FORSE) è più complesso perché richiederebbe funzionalità avanzate come il service discovery, DNS routing ecc.
* È necessario un team più ampio per mantenere l’applicazione in quanto vi sono più component e più tecnologie coinvolte.
* I membri del team usano tecnologie diversi in base al componente su cui lavorano e questo rende più difficile la condivisione delle conoscenze.
* Lo stack tecnico è complesso e spesso difficile da imparare.
* Il tempo di sviluppo iniziale sarà più alto, rendendo più lenta la commercializzazione.
* Richiede un infrastruttura più complessa rispetto alle applicazioni monolitiche. Più spesso saranno richiesti container e multiple JVM per l’esecuzione.

QUANDO PREFERIRE UN APP MONOLITICA:

Si preferisce un app monolitica quando lo scope dell’applicazione è piccolo e ben definito (un blog, un piccolo sito di shopping online, una semplice CRUD application, ecc.), quando il team è composto da poche persone (circa 8 per esempio), quando la media delle skill del team è pari al novizio o all’intermedio, quando il tempo di commercializzazione è breve, quando non si vogliono spese eccessive sull’infrastruttura oppure quando l’utente che userà il sistema è piccolo e non ti aspetti che cresca troppo.

QUANDO PREFERIRE UN APP A MICROSERVIZI:

Quando lo scope dell’applicazione è grande e ben definito e si è sicuri che l’applicazione crescerà tremendamente in termini di features (per esempio uno store e-commerce online, un social media, un servizio di video streaming con una larga base di utenti, un API provider ecc., quando ci sono abbastanza membri per sviluppare i singoli componenti in maniera indipendente, quando le skill media del team sono buone e i membri sono confidenti riguardo i pattern avanzati relativi ai microservizi, quando il tempo di commercializzazione non è critico visto che l’architettura a microservizi prenderà più tempo per essere sviluppata, quando si ha la possibilità di spendere di più in termini di infrastrutture, monitoraggio ecc. per migliorare la qualità del prodotto, quando la base di utenza è alta e ti aspetti che continui a crescere (per esempio un social media dove l’utenza target è tutto il mondo).