Jakarta EE, DevOps, PlatformEng

https://jakarta.ee/

https://platformengineering.org/

Jakarta EE, DevOps, PlatformEng

- Vedremo perché è nata la PlatformEng e quali soluzioni porta alla DevOps
- Successivamente si vedrà come jakarta ee si cala perfettamente nella nuova disciplina PlatformEng.

Questo lavoro si pone l'obiettivo di portare e diffondere un breve un sunto del perchè è nata la metodologia/disciplina della PlatformEng come evoluzione della DevOps....

In particolare in aiuto degli sviluppatori e del business....

Limiti di DevOps e la Nascita di Platform Engineering

JAKARTA EE

Limiti di DevOps e la Nascita di Platform Engineering

DevOps ha rivoluzionato il modo in cui software viene sviluppato e rilasciato, abbattendo i silos tra Sviluppo (Dev) e Operations (Ops) e promuovendo collaborazione, automazione (CI/CD) e feedback rapido. Tuttavia, specialmente in organizzazioni/business grandi o complesse, l'implementazione di DevOps può incontrare alcune sfide:

IN PROGETTI/BUSINESS DELICATI E COMPLESSI L'INTRODUZIONE DELLA SOLA E PURA DevOps a portato a sovraccaricare il team di sviluppo che dovrebbe concentrarsi al 90% solo sullo sviluppo del business.

In pratica, un team di sviluppo nel mondo DevOps viene tirato in causa per ogni esigenza non funzionale, anzichè essere concentrato sulle evolutive/gestione del business/prodotto.

Platform Engineering: SCRIVERE CODICE DI VALORE

Platform Engineering è emersa come risposta alla crescente complessità dello sviluppo software moderno, in particolare con l'adozione del cloud computing, dei microservizi e delle pratiche DevOps. I team di sviluppo si trovavano spesso a dover gestire infrastrutture complesse, configurare ambienti, integrare tool diversi e risolvere problemi operativi, distraendosi dal loro obiettivo principale: scrivere codice di valore.

In pratica, un team dedicato (il Platform Team) crea e mantiene una piattaforma integrata che astrae la complessità dell'infrastruttura sottostante e delle operazioni. I team di sviluppo possono quindi utilizzare questa piattaforma tramite interfacce chiare e modalità self-service per costruire, testare, rilasciare e operare le proprie applicazioni in modo rapido, sicuro e conforme agli standard aziendali, senza doversi necessariamente preoccupare di tutti i dettagli implementativi sottostanti.

Limiti di DevOps e la Nascita di Platform Engineering

Il Platform Engineering è una disciplina emergente che si concentra sulla progettazione, costruzione e gestione di piattaforme interne self-service destinate ai team di sviluppo software all'interno di un'organizzazione. L'obiettivo principale è migliorare l'esperienza dello sviluppatore (Developer Experience) e la produttività, fornendo loro strumenti, servizi, infrastrutture e workflow automatizzati e standardizzati – spesso chiamati "Golden Paths" o "Percorsi Abilitati".

In pratica, un team dedicato (il Platform Team) crea e mantiene una piattaforma integrata che astrae la complessità dell'infrastruttura sottostante e delle operazioni. I team di sviluppo possono quindi utilizzare questa piattaforma tramite interfacce chiare e modalità self-service per costruire, testare, rilasciare e operare le proprie applicazioni in modo rapido, sicuro e conforme agli standard aziendali, senza doversi necessariamente preoccupare di tutti i dettagli implementativi sottostanti.

L'obiettivo principale del Platform Engineering

L'obiettivo principale del Platform Engineering è migliorare l'esperienza degli sviluppatori (Developer Experience - DX) e aumentare l'efficienza dei team di sviluppo.

JAKARTA EE

Come i team di Platform Engineering sfruttano il modello Jakarta EE per fornire un runtime finito:

- **Astrazione dalla complessità:** Nasconde la complessità dell'infrastruttura e delle operazioni, offrendo interfacce semplici e self-service.
- **Standardizzazione e coerenza:** Promuove l'utilizzo di tool e processi standardizzati, garantendo maggiore coerenza e affidabilità.
- **Automazione:** Automatizza task ripetitivi e manuali, liberando tempo per attività più strategiche.
- **Self-service:** Permette agli sviluppatori di accedere alle risorse e ai servizi di cui hanno bisogno in autonomia, senza dover dipendere da altri team.
- **Governance e sicurezza:** Implementa meccanismi di governance e sicurezza in modo centralizzato, garantendo la conformità e la protezione dei dati.

JAKARTA EE

Standard e Governance più Spinta:

Jakarta EE: Si basa su specifiche standard. Una piattaforma basata su Jakarta EE può fornire runtime (Application Server come Glassfish, Weblogic, WildFly, Open Liberty, Payara etc) pre-configurati, securizzati e ottimizzati dal team della piattaforma. I team di sviluppo si concentrano sulla logica di business aderendo a queste API standard (JPA, JAX-RS, CDI, etc.).

Selezione e Preconfigurazione del Server Applicativo e della sua Container Image: Grazie al modello Jakarta EE, E' il team di Platform Engineering che può scegliere un server applicativo (ad esempio, WildFly, GlassFish, Payara) che meglio si adatta alle esigenze dell'organizzazione e **preconfigurarlo** con impostazioni ottimali per sicurezza, prestazioni e conformità. Questo server applicativo preconfigurato diventa il "runtime finito" offerto dalla piattaforma e allo sviluppo..

Contrasto con Immagine Docker con Uber JAR in DevOps: Ogni uber JAR e la sua immagine docker è responsabilita' del team di sviluppo, porta con sé le *proprie* dipendenze e il *proprio* server embedded (Tomcat, Jetty, Undertow). Questo può portare a una proliferazione di versioni diverse dello stesso componente (server, librerie) nell'ecosistema, rendendo più complessa la governance centralizzata e l'applicazione di patch di sicurezza uniformi a livello di runtime.

Vantaggio PE: Il team della piattaforma definisce e gestisce un insieme limitato di runtime standardizzati, garantendo coerenza e semplificando la manutenzione e la sicurezza a livello infrastrutturale.

Separazione delle Competenze (Runtime vs Sviluppo Applicazione Business):

Jakarta EE: C'è una chiara separazione tra l'ambiente di runtime (l'Application Server, gestito dal team della piattaforma) e l'artefatto applicativo (WAR/EAR, sviluppato dal team di sviluppo).

Contrasto con Uber JAR: L'uber JAR fonde runtime e applicazione. Il team di sviluppo è responsabile dell'intero pacchetto, inclusa la configurazione e le dipendenze del server embedded.

Vantaggio PE: Questa separazione si allinea bene con il modello Platform Engineering. Il team della piattaforma si occupa dell'infrastruttura e del runtime (tuning, patching, scaling dell'Application Server), mentre i team di sviluppo si concentrano sul codice applicativo, interagendo con la piattaforma tramite interfacce definite (deployment su server gestito).

Consistenza Operativa e Gestione Centralizzata:

Jakarta EE: Le operazioni (monitoring, logging, patching, aggiornamenti di sicurezza del runtime) vengono eseguite sull'Application Server in modo centralizzato dal team della piattaforma e si applicano a tutte le applicazioni deployate su di esso.

Contrasto con Uber JAR: La gestione operativa è distribuita. Ogni applicazione/team potrebbe richiedere strategie leggermente diverse per logging, monitoring, e l'aggiornamento delle dipendenze (incluso il server embedded) richiede una nuova build e deployment dell'intero uber JAR.

Vantaggio PE: Semplifica la vita del team della piattaforma, che può applicare best practice operative e di sicurezza in modo uniforme su un insieme noto di Application Server gestiti.

THANK YOU!