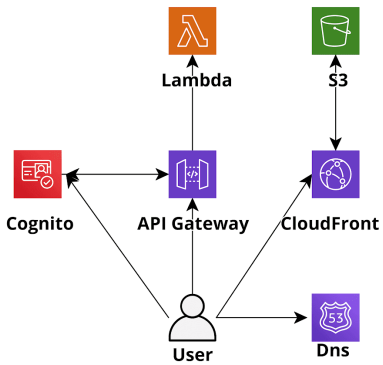
CREARE E MANTENERE UN'INFRASTRUTTURA AWS SERVERLESS CON TROPOSPHERE E CODEPIPELINE.

AWS CodePipeline | Infrastructure as Code (IaC) | Serverless | Troposphere

beSharp | 6 Marzo 2020

Le architetture AWS serverless, nella maggior parte dei casi di uso, offrono enormi vantaggi rispetto alle loro controparti "classiche" sviluppate su istanze EC2.

Prendiamo in considerazione, a titolo di esempio, una semplice web application serverless sviluppata usando AWS Lambda (backend), DynamoDB (database), Cognito (Authentication) e S3 – CloudFront (Angular single page application): un'applicazione sviluppata in questo modo sarà in grado di scalare molto rapidamente, adattandosi in modo quasi istantaneo a qualunque livello di traffico. Inoltre costerà significativamente meno di un applicazione con hosting su EC2 visto che verrà fatturato da AWS un prezzo proporzionale al traffico effettivamente ricevuto invece del costo fisso di una macchina virtuale sempre accesa. Un ulteriore vantaggio consiste nell'abbattimento dei costi di manutenzione dato che i sistemisti non dovranno più occuparsi di aggiornare il sistema operativo, configurare l'hardening della macchina, installare patch di sicurezza etc.



Tutto ciò fa sì che migrare verso il paradigma serverless sia una scelta naturale in buona parte delle situazioni: si andrà a pagare meno per un servizio più scalabile e più manutenibile.

Tuttavia il vero prezzo da pagare per usufruire per usufruire dei vantaggi del paradigma serverless è l'aumento della complessità dell'infrastruttura AWS: mentre con una semplice applicazione deployata su macchine EC2 è sufficiente creare un AMI della configurazione funzionante del

sistema operativo, agganciarla ad un autoscaling group e configurare un Load balancer per essere pronti ad andare in produzione, nel caso di un'applicazione serverless le componenti architetturali da configurare sono molteplici. Di solito si dovranno gestire svariate Lambda, spesso un numero paragonabile di rotte su Api Gateway, oltre ovviamente alla configurazioni di DynamoDB. Inoltre a differenza del codice deployato su EC2, che è di solito autocontenuto, il codice della lambda è spesso fortemente integrato con l'infrastruttura e modifiche al codice spesso richiedono di modificare anche l'infrastruttura Per esempio l'aggiunta di una nuova Api tipicamente richiede la creazione di una nuova funzione lambda e della corrispondente rotta su Api Gateway.

Ovviamente cercare di gestire manualmente tutte queste risorse diventa rapidamente impossibile e rende necessario ricorrere ad un diverso approccio: gestire le varie risorse, infrastruttura compresa, tramite pipelines di Continuous Integration/Continuous Delivery (CD/CI) con AWS CodePipeline. Nel caso del backend, in questo tipo di set-up, ogni volta che un developer esegue un push sul repo Git, il codice viene scaricato dal repo direttamente da AWS CodePipeline, viene eseguita una build del codice tramite AWS CodeBuild che prepara un pacchetto (o dei pacchetti) contenenti il codice delle Lambda e anche uno o più template Cloudformation che descrivono i cambiamenti che devono essere apportati all'infrastruttura per consentire il funzionamento delle nuove Lambda. Infine, nell'ultimo step della pipeline, Cloudformation si occupa di eseguire il template aggiornando/creando le lambda e allo stesso tempo creando/modificando le varie risorse infrastrutturali usate dalle lambda.

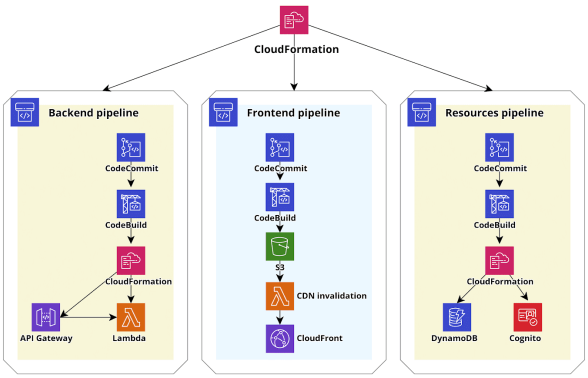
Sebbene sia possibile gestire l'intera infrastruttura con un solo Cloudformation/Pipeline questo risulta spesso complesso e poco efficiente, pertanto dovrebbe essere evitato. Un modo migliore di procedere è invece quello di **creare più Stacks Cloudformation,** uno per ogni macro componente dell'applicazione. Per esempio nel caso piuttosto semplice di una web application classica come quello presentato all'inizio del post si potrebbe creare tre Cloudformation stacks, uno per il backend, uno per frontend ed uno per gli altri componenti infrastrutturali (Dynamo, Cognito, etc) ognuno deployato in modo automatico da una sua pipeline dedicata.

La creazione di una pipeline dedicata solamente alle varie risorse necessarie all'infrastruttura serverless, ma non strettamente legate al codice, consente agli sviluppatori di avere una grossa libertà di movimento in quanto possono autonomamente creare e testare nuove funzionalità ed eseguire modifiche e miglioramenti all'infrastruttura dell'Account AWS senza avere la possibilità di eseguire azioni distruttive.

Per semplificare la vita dei developer sarebbe consigliabile usare Troposphere come linguaggio di Infrastructure as Code e trasformare i file python di troposphere in template YAML solo nei vari step di Build. In questo modo è possibile usufruire della versatilità di CloudFormation unita alla semplicità di un linguaggio ad alto livello interpretato come Python.

Inoltre sarebbe preferibile che tutta la configurazione base dell'account, come la configurazione di CloudTrail, dei FlowLogs della VPC e la creazione delle pipeline sopra descritte venisse portata a termine da un ulteriore Cloudformation Stack dedicato in modo da avere una "ricetta" base che possa essere utilizzata come documentazione e che possa rendere possibile se necessario la

configurazione rapida di cccount identici, per esempio destinati ad altri ambienti di sviluppo o a necessità di disaster recovery.



Il Cloudformation Template per la parte di Backend dell'applicazione può essere creato direttamente dagli sviluppatori oppure generato tramite una routine automatica da uno dei tanti framework per sviluppo Serverless come il Serverless Framework, chalice, Zappa o da AWS Amplify. In caso si richieda una maggiore versatilità è anche possibile usare un SAM. AWS SAM è un linguaggio di templating semplificato rispetto a cloudformation che consente di creare le varie risorse tipiche dei progetti serverless (e.g. funzioni Lambda, Api gateways etc..). Un template SAM può poi essere convertito direttamente in un template cloudformation tramite un comando della AWS cli (AWS CloudFormation package).

Per concludere abbiamo presentato una procedura per rendere facilmente mantenibile una applicazione serverless anche molto complessa utilizzando CodePipeline e Troposphere/Cloudformation. Se siete curiosi e volete saperne di più non esitate a scriverci nei commenti o per mail!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it