Come arricchire le pipeline di CD/CI con la static code analysis utilizzando SonarQube

25 Giugno 2021 - 9 min. read

Continuous Delivery | Continuous Integration

Abbiamo più volte trattato le pipeline di continuous delivery nel corso degli anni; questo perché crediamo davvero che disporre di un processo automatico, affidabile e completamente gestito per testare e distribuire il codice aiuti ad aumentare la qualità e la quantità del codice prodotto.

Configurare una buona pipeline di CI/CD è un aspetto fondamentale per accelerare la distribuzione del software senza sacrificarne la qualità, inoltre è imprescindibile includere uno step di static code analysis nelle pipeline per sfruttarne a pieno i vantaggi e le potenzialità.

Uno strumento di static code analysis ispeziona la codebase durante il ciclo di sviluppo ed è in grado di identificare bug, vulnerabilità e problemi di conformità senza eseguire effettivamente il codice sorgente.

L'analisi del codice può aiutare a garantire che il software sia sicuro, affidabile e conforme agli standard di qualità e stile definiti a livello aziendale.

Che cos'è la static code analysis?

La static code analysis è una pratica che consente ai team di sviluppo di rilevare automaticamente potenziali bug, problemi di sicurezza e, più in generale, difetti nel codice sorgente di una o più applicazioni. Pertanto, possiamo vedere l'analisi statica come un ulteriore processo di revisione del codice automatizzato. Esaminiamo questa analogia più in dettaglio.

Il processo di revisione del codice, o code review, è probabilmente il modo migliore per garantire la qualità del codice prodotto da un team di sviluppo. Durante una code review, una coppia di sviluppatori esamina il codice con l'obiettivo preciso di migliorarlo e di individuare pratiche pericolose sia dal punto di vista della manutenibilità che della sicurezza.

Durante il processo di revisione, l'autore del codice non dovrebbe spiegare come funzionano determinate parti del programma in modo che il revisore non sia prevenuto sul suo giudizio. Inoltre, il codice dovrebbe essere chiaro, semplice, e altamente gestibile; la complessità dovrebbe essere mitigata dall'astrazione e dall'incapsulamento. Infine, il codice dovrebbe essere ritenuto sufficientemente chiaro, gestibile e sicuro, da entrambi i programmatori per superare la revisione.

Le code review sono particolarmente efficaci perché è più facile per uno sviluppatore, individuare bug, code smell, e suggerire miglioramenti al codice prodotto da qualcun altro, piuttosto che sul suo lavoro, per il quale si ha un forte bias.

La pratica delle review dovrebbe essere eseguita con la maggiore frequenza possibile; tuttavia, l'attività richiede molto tempo e di conseguenza risulta molto costosa sia in termini di tempo che di denaro.

Un modo eccellente per aumentare la frequenza delle revisioni del codice consiste quindi nell'includere uno step di code review automatica all'interno delle pipeline utilizzando uno strumento di static code analysis.

Strumenti e soluzioni di static code analysis

Esistono strumenti e soluzioni per implementare la static code analysis in grado di scansionare automaticamente la codebase e generare report accurati sulle condizioni del codice che sono rivolti agli sviluppatori. Tali strumenti sono solitamente facili da integrare nelle pipeline CD/CI; solitamente il comando di scansione restituisce un exit code mediante il quale è possibile determinare se il codice è sufficientemente buono o se non supera l'analisi statica.

Naturalmente, una soluzione completamente automatizzata non può sostituire una revisione completa del codice eseguita da uno o più sviluppatori. Tuttavia, l'aumento del rapporto tra la frequenza dell'analisi del codice e l'impatto relativamente economico sul prezzo complessivo rende l'aggiunta di uno step di analisi alla pipeline un modo efficiente per migliorare la qualità e la sicurezza del codice.

Aggiungere questo step non sostituisce le code review, ma la sua aggiunta porta sicuramente benefici a buon mercato.

- Rilevazione di possibili bug e problemi di sicurezza
- Consigli sulla formattazione del codice e rilevazione di numerose tipologie di code smell
- Calcolo di code metrics

Molti strumenti di code analysis, sia commerciali che gratuiti, supportano una vasta pletora di linguaggi di programmazione. Uno dei più famosi è **SonarQube**, che descriveremo meglio in seguito.

Direttamente da AWS possiamo anche sfruttare **CodeGuru**, un servizio basato sul machine learning, che è facile da integrare nelle pipeline e può fornire suggerimenti di alta qualità per migliorare il codice. Sfortunatamente, al momento, CodeGuru supporta solo Java e Python, quest'ultimo in preview.

CodeGuru mira a diventare uno strumento di analisi robusto e di alta qualità. Tuttavia, al momento, è stabile da utilizzare solo per gli sviluppatori Java; quindi, ci concentreremo su una soluzione più matura che può essere utilizzata per molte più linguaggi.

Questo articolo approfondirà come integrare **SonarQube** in una pipeline di CD/CI.

SonarQube è un software open source per il controllo continuo della qualità del codice. Esegue revisioni automatiche mediante analisi statica e supporta più di 20 linguaggi di programmazione. Può individuare il codice duplicato, calcolare la copertura del codice, la complessità del codice e trovare bug e vulnerabilità di sicurezza. Inoltre, può registrare la cronologia delle metriche e fornisce grafici di evoluzione tramite un'interfaccia web dedicata.

Lo svantaggio dell'utilizzo di SonarQube è che per utilizzarlo occorre abbonarsi ad un servizio sonarqube gestito (non fornito da AWS) o gestire un cluster con la propria installazione.

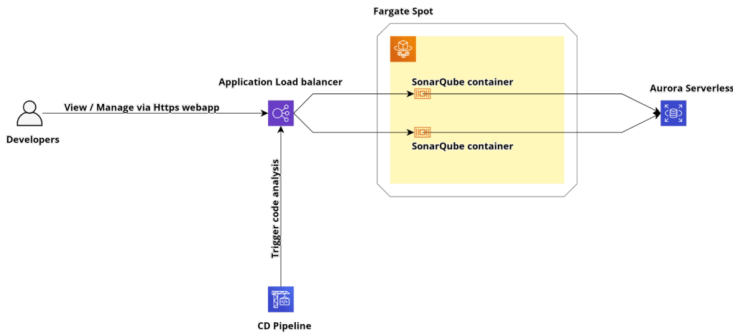
Di solito tendiamo a consigliare e sfruttare servizi fully managed per la pipeline di sviluppo, perché consentono di concentrarsi sul core business piuttosto che sull'infrastruttura o sugli strumenti necessari per realizzare la pipeline.

Tuttavia, in questo caso, abbiamo optato per una soluzione personalizzata a causa di un modello di pricing non compatibile con il nostro utilizzo dello strumento. Inoltre, la fase di revisione automatica non è un bloccante per la pipeline durante lo sviluppo, rendendo la disponibilità del cluster non critica.

Setup di SonarQube su AWS

In questo breve tutorial illustreremo ad alto livello come configurare un cluster SonarQube su AWS sfruttando i servizi gestiti.

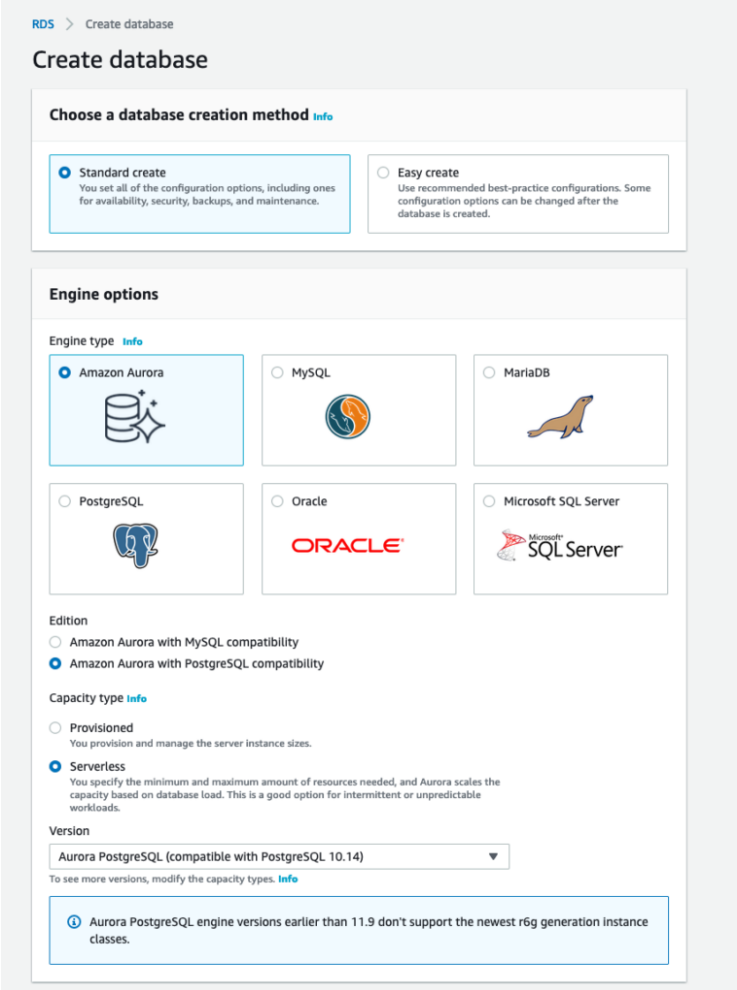
Realizzeremo un cluster altamente disponibile e scalabile basato su ECS Fargate e Amazon Aurora Serverless



Creazione del cluster Amazon Aurora Serverless

Per funzionare correttamente, SonarQube necessita di un database PostgreSQL o MySQL. In questo tutorial utilizzeremo Amazon Aurora Serverless con compatibilità PostgreSQL.

Per creare il database basta accedere alla Console di gestione AWS, navigare sino alla sezione AWS RDS e fai cliccare sul pulsante "Crea database". Nel modulo seguente, selezionare Amazon Aurora come engine, Amazon Aurora con compatibilità PostgreSQL come edizione e Serverless come tipo di capacità. Un riassunto delle scelte è disponibile nell'immagine qui sotto.



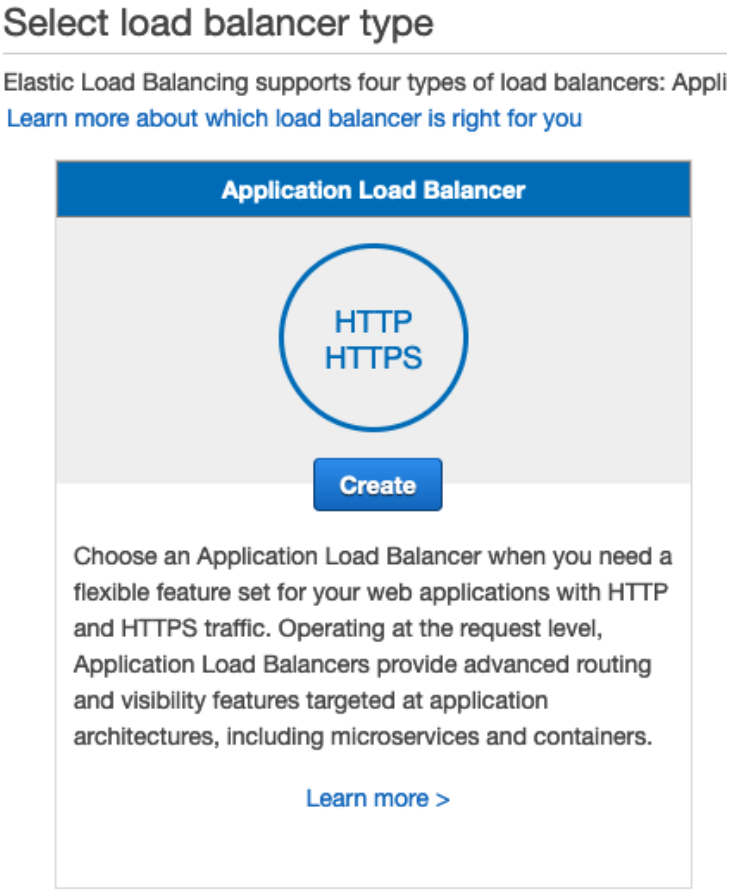
Infine, completare la configurazione del cluster impostando il nome del database, la password e tutta la configurazione di rete.

Creazione di un Application Load Balancer e del Target Group

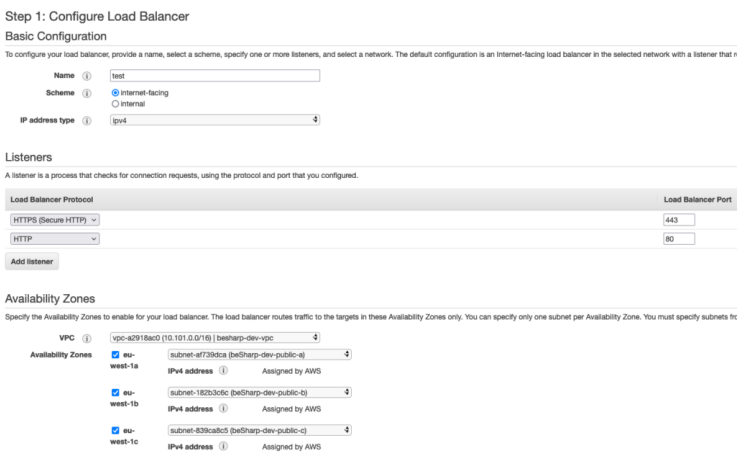
Per esporre il servizio Fargate che conterrà la nostra applicazione Sonarqube, dobbiamo creare un AWS Application Load Balancer con il suo target group. Per abilitare HTTPS, occorre creare o importare un certificato SSL all'interno di AWS Certificate Manager. In caso contrario, nella creazione del load balancer, è possibile configurare solo il listener sulla porta 80.

Iniziamo quindi a creare il target group dalla pagina del servizio EC2. Nella sezione target group fare clic sul pulsante "Crea target group". Selezionare "Indirizzo IP" come tipo di destinazione, HTTP come protocollo e 9000 come porta; assicurarsi inoltre di selezionare HTTP1 come versione del protocollo.

Ora che abbiamo il nostro target group, possiamo creare l'Application Load Balancer. Per farlo, dalla sezione Load Balancer all'interno della Console EC2 fare clic sul pulsante Crea Load Balancer. Quindi, fare clic sul pulsante Crea nella sezione Application Load Balancer nella procedura guidata come nell'immagine sottostante.

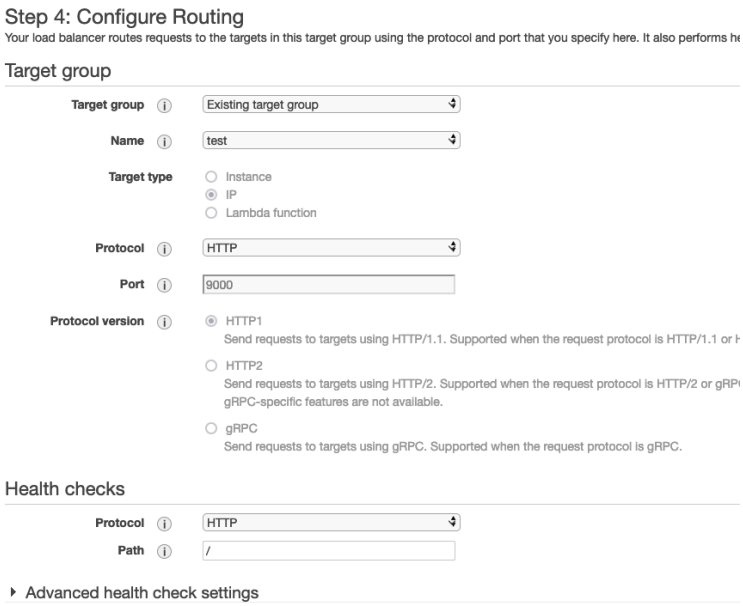


Selezionare il tipo internet facing e creare due listener, uno sulla porta 80 e uno sulla porta 443. Il bilanciatore va associato alle subnet pubbliche per rendere il cluster raggiungibile da internet.

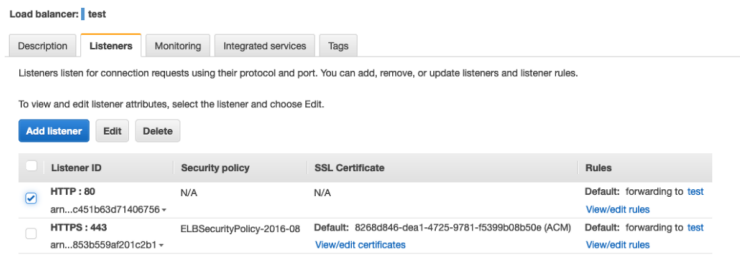


Nella sezione successiva (solo se si crea il listener sulla porta 443), seleziona un certificato SSL da AWS Certificate Manager per abilitare HTTPS.

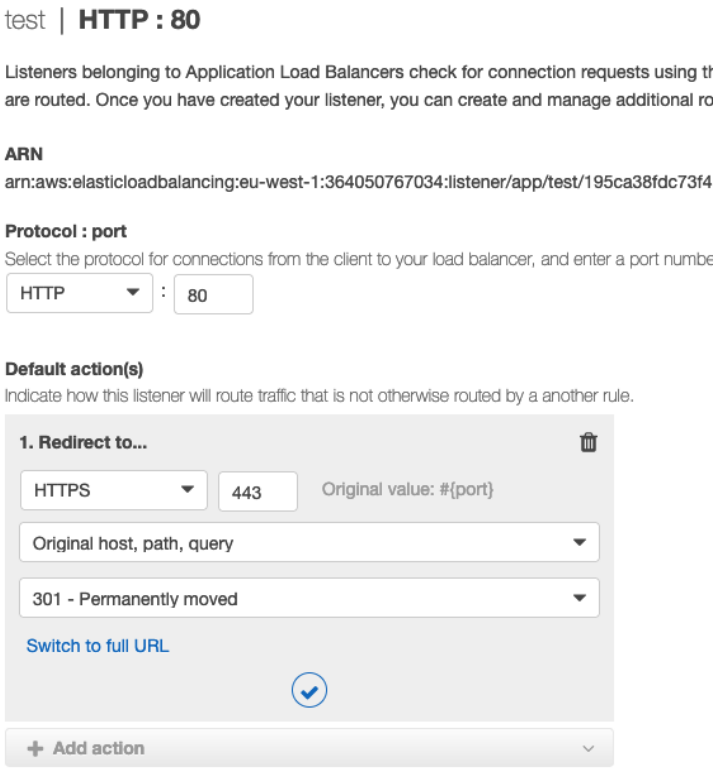
Infine, selezionare il target group creato precedentemente.



Se si sceglie di esporre il cluster via HTTPS occorre modificare il comportamento della regola di default per implementare un redirect da http ad https e inoltrare il traffico https ai container fargate. Nelle immagini qui sotto sono visibili i passaggi necessari.

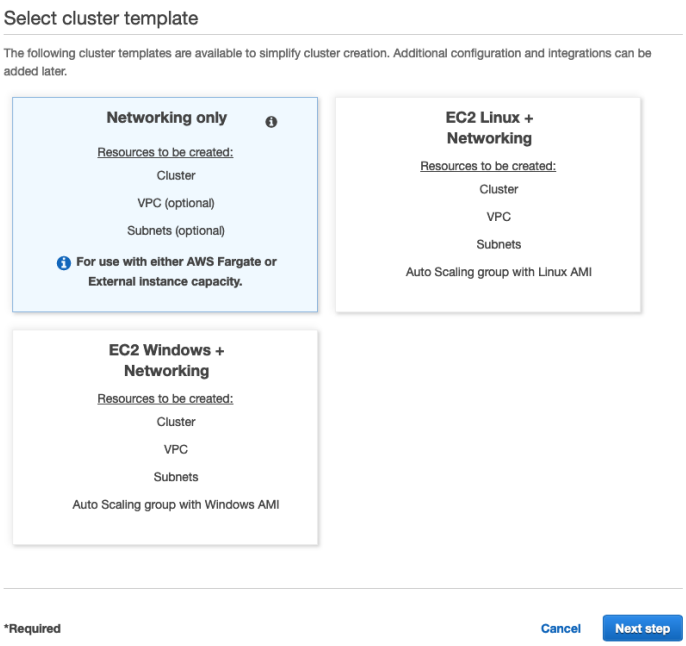


Nella pagina di modifica, eliminare il comportamento predefinito e crearne uno nuovo facendo clic sul pulsante "Aggiungi azione". Nella lista di controllo, selezionare il valore "Reindirizza a" per reindirizzare il traffico http al listener https.



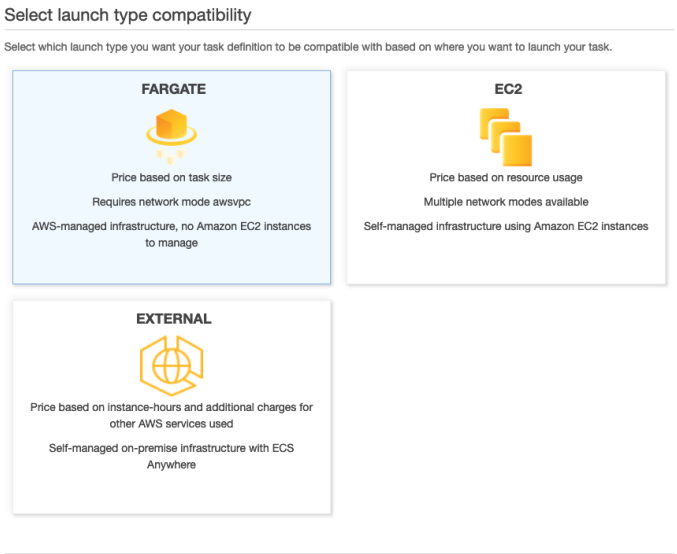
Creare il Cluster Fargate

Dalla console di ECS nella sezione Cluster ECS fare clic sul pulsante "Crea cluster". Quindi, selezionare il modello "Networking only" come l'immagine qui sotto.

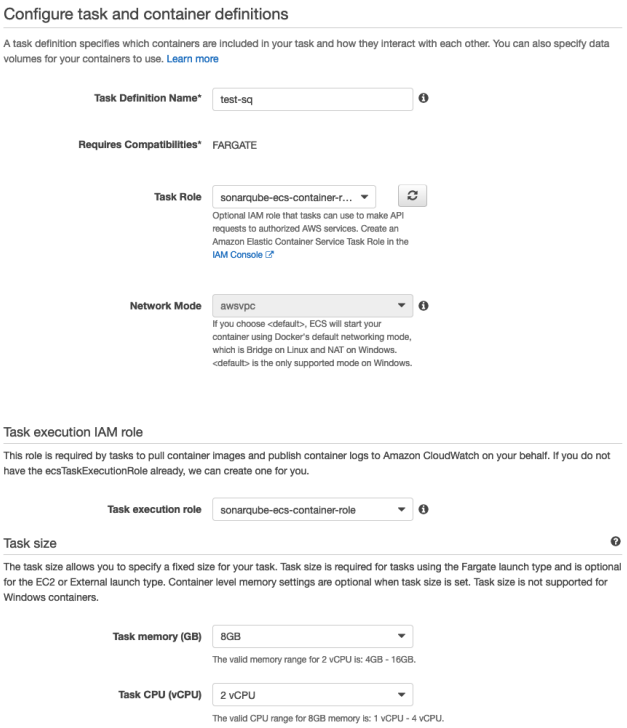


Indicare un nome per il cluster e procedere con il wizard.
Dal pannello di gestione IAM, creare un nuovo ruolo IAM e collegarvi le policy gestite denominate "AmazonECSTaskExecutionRolePolicy" e "AmazonEC2ContainerServiceRole ".

Nella sezione successiva selezionare "Crea nuova task definition" e selezionare il tipo Fargate.

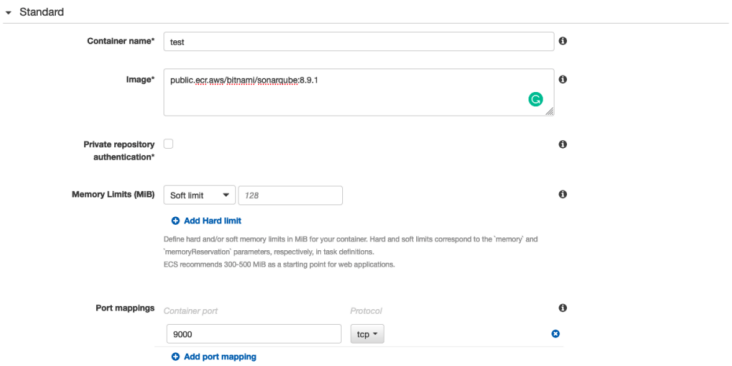


Impostare quindi il ruolo IAM precedentemente configurato e selezionare il dimensionamento dei container a 8 GB per la RAM e 2 vCPU.



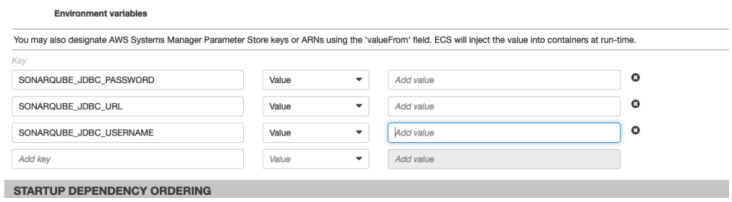
Nella sezione container, aggiungere un nuovo container ed utilizzare "public.ecr.aws/bitnami/sonarqube:8.9.1" come immagine. Quest'immagine è la versione ufficiale di Sonarqube ospitata da AWS su un ECR pubblico. Volendo, è possibile utilizzare il proprio repository privato ECR con una versione modificata o

comunque controllata dell'immagine di SonarQube. Nella mappatura delle porte, occorre mappare la porta 9000 del container.



Nella sezione delle variabili di ambiente occorre impostare i seguenti valori per il corretto funzionamento dell'immagine di SonarQube:

- SONARQUBE_JDBC_URL: La stringa di connessione jdbc per la connessione al cluster Aurora Serverless come da seguente template: *jdbc:postgresql:/**YOUR_DATABASE_ENDPOINT**:5432/**YOUR_DATABASE_NAME?sslmode=require&gssEncMode=disable***
- SONARQUBE_JDBC_USERNAME: Username del database aurora
- SONARQUBE_JDBC_PASSWORD> Password dell'utente del db



A questo punto, è possibile configurare un servizio per il cluster SonarQube. Il servizio è la definizione di un'attività e un insieme di parametri che determinano quante istanze dell'attività sono necessarie. Abbiamo parlato di come configurare i servizi per ECS in altri articoli, per esempio qui.

Alla fine, ECS avvierà le istanze necessarie e sarà possibile accedere al software mediante l'URL del load balancer.

Come integrare la scansione SonarQube nella pipeline

Ora che il cluster è attivo e funzionante, possiamo accedervi e iniziare a configurare SonarQube. È possibile mettere a punto la configurazione e le preferenze delle scansioni. Una volta che il nostro progetto è stato creato e configurato, possiamo configurare l'automatismo per analizzare il codice durante l'esecuzione della pipeline.

Ci sono diversi modi per raggiungere questo obiettivo. Il più comune e facile da implementare è semplicemente aggiungere l'esecuzione di un comando alla fase di build su CodeBuild. Per avviare un'analisi, è necessario installare un agent e quindi eseguire un comando per avviare il processo.

L'agent può essere preinstallato nel contenitore di build, l'ultima versione è disponibile per il download qui.

Quando l'agent è a posto, è possibile avviare un'analisi eseguendo questo comando.

sonar-scanner -Dsonar.projectKey=[PROJECT_KEY]-Dsonar.sources=.-Dsonar.host.url=[LOAD_BALANCER_URL]-Dsonar.login=[LOGIN_KEY]-Dsonar.qualitygate.wait=true-Dsonar.qualitygate.wait=true
L'ultimo parametro indica allo scanner di attendere la fine della scansione e di restituire un exit code diverso da 0 se la qualità del rilascio è inferiore alla soglia configurata.

In questo modo, la fase di compilazione della pipeline fallirà se la qualità del codice non è aderente alle regole definite su SonarQube o se sono stati individuati bug o pratiche pericolose.

Capire quando è il caso di far fallire la pipeline se la qualità non è sufficiente è un aspetto cruciale dell'intero processo.

Come già suggerito, l'analisi del codice è economica e dovrebbe essere eseguita il più possibile. Tuttavia, il processo di sviluppo non dovrebbe essere interrotto ogni volta che un'analisi del codice fallisce a seconda dell'ambiente in cui si sta effettuando il rilascio.

È utile scansionare e generare il report ad ogni commit senza interrompere il rilascio, specialmente negli ambienti di sviluppo.

Le pipeline che afferiscono a qualsiasi ambiente "non di sviluppo" dovrebbero fallire se l'analisi del codice non è sufficientemente buona, utilizzando il parametro evidenziato sopra.

Conclusioni

La static code analysis è una pratica affidabile e preziosa, che merita certamente di essere inclusa nel ciclo di sviluppo.

Esistono opzioni AWS come CodeGuru, completamente gestite e basate su Machine learning, ed esistono anche molte piattaforme commerciali e/o open source come SonarQube che possono fornire supporto per i linguaggi specifici utilizzati dai team di sviluppo.

Indipendentemente dall'ambito e dalla tipologia di applicazione, se si dispone di pipeline di CD/CI è bene considerare l'aggiunta di una fase di analisi automatica del codice e assicurarsi che il rilascio si interrompa in caso la qualità non risulti sufficiente, almeno per gli ambienti di produzione.

Restate sintonizzati per altri articoli sulle pipeline di continuous delivery e su altri approfondimenti sull'analisi del codice sorgente.

Ci vediamo tra 14 giorni qui, su **#Proud2beCloud**!

Alessio Gandini

Cloud-native Development Line Manager @ beSharp, DevOps Engineer e AWS expert.Computer geek da quando avevo 6 anni, appassionato di informatica ed elettronica a tutto tondo. Ultimamente sto esplorando l'esperienza utente vocale e il mondo dell'IoT.Appassionato di cinema e grande consumatore di serie TV, videogiocatore della domenica.

Simone Merlini

CEO e co-fondatore di beSharp, Cloud Ninja ed early adopter di qualsiasi tipo di soluzione *aaS. Mi divido tra la tastiera del PC e quella a tasti bianchi e neri; sono specializzato nel deploy di cene pantagrueliche e nel test di bottiglie d'annata.
