
Università degli Studi di Padova

Dipartimento di Matematica "Tullio Levi-Civita"

Corso di Laurea in
Informatica


# Studio, progettazione e sviluppo di infrastrutture programmabili per il cloud computing su tecnologie AWS


*Relatore*
Prof. Davide Bresolin

*Laureando*
Davide Spada
1220539



Anno Accademico 2021/2022

# Sommario

Il progetto di stage intitolato “Infrastructure as Code, what else” ruota attorno alla programmabilità delle infrastrutture cloud tramite codice. Sempre di più le infrastrutture sono progettate in modo integrato con il software applicativo che dovrà essere realizzato. I vantaggi di definire infrastrutture in tale modo, impiegando servizi cloud come gli *Amazon Web Services* offerti da Amazon, sono molteplici e verranno trattati nelle sezioni dedicate a seguire.
L’obiettivo del progetto proposto da Zero12 è quello di realizzare template di infrastrutture tramite codice per diverse tipologie di workload, come Web applications, servizi serverless o applicazioni containerizzate, integrati con sistemi di integrazione e consegna continua (CI/CD). La forma a template indica che le infrastrutture sviluppate e quindi i suoi componenti, hanno strutture definite a priori per soddisfare determinati requisiti di progettazione, ma consentono ai singoli componenti di assumere proprietà intrinseche diverse in base alle configurazioni modificabili tramite un opportuno file di configurazione.
La progettazione e lo sviluppo delle infrastrutture è stato realizzato tramite le tecnologie AWS a seguito di uno studio preliminare che ha occupato una buona parte dello stage. In particolare i servizi studiati sono stati integrati nello sviluppo di infrastruttura templatizzate in tre principali ambiti impiegati anche in produzione: pipeline, serverless e container.

# Indice


**1 Introduzione** **1**

1.1 Azienda . . . . . 1

1.2 Progetto . . . . . 3

**2 Introduzione ad AWS** **5**

2.1 Cloud computing . . . . . 5

2.2 Amazon Web Services . . . . . 7

2.3 Console AWS . . . . . 8

2.4 AWS CloudFormation . . . . . 9

**3 Servizi e Tecnologie fondamentali per lo sviluppo AWS** **11**

3.1 Servizi principali per lo sviluppo di infrastrutture cloud . . . . . 11

3.1.1 Amazon IAM . . . . . 12

3.1.2 Amazon VPC . . . . . 14

3.1.3 Amazon EC2 . . . . . 16

3.1.4 Amazon RDS . . . . . 17

3.1.5 Amazon S3 . . . . . 18

3.2 Tecnologie per lo sviluppo di infrastrutture AWS . . . . . 19

3.2.1 AWS CLI . . . . . 19

3.2.2 AWS CDK . . . . . 20

3.2.3 Node JS & npm . . . . . 21

3.2.4 TypeScript . . . . . 22

3.2.5 Visual Studio Code . . . . . 23

**4 Progettazione e Sviluppo di infrastrutture AWS** **25**

4.1 Concetti chiave . . . . . 25

4.2 Inizializzazione di un'app in CDK . . . . . 28

4.3 Configurazione del template di progetto . . . . . 29

**5 Template di infrastruttura con servizio Pipeline** **31**

5.1 Amazon VPC . . . . . 32

5.2 Amazon EC2 Auto Scaling (ASG) . . . . . 33

5.3 Amazon Application Load Balancer (ALB) . . . . . 34
5.4 Amazon Relational Database Service (RDS) . . . . . 35
5.5 Amazon CodePipeline . . . . . 36

**6 Template di infrastruttura con servizio Serverless 39**
6.1 Amazon CloudFront . . . . . 40
6.2 Amazon Cognito . . . . . 41
6.3 Amazon API Gateway . . . . . 42
6.4 Amazon Lambda . . . . . 44
6.5 Amazon DynamoDB . . . . . 46

**7 Template di infrastruttura con servizio Container 47**
7.1 Docker e Kubernetes . . . . . 49
7.2 Amazon EKS . . . . . 51
7.2.1 Introduzione . . . . . 51
7.2.2 Worker nodes e Amazon Fargate . . . . . 52
7.2.3 Componenti Kubernetes . . . . . 52
7.3 Amazon Aurora . . . . . 53

**8 Distribuzione 55**
8.1 Bootstrapping . . . . . 55
8.2 Confronto di versioni . . . . . 56
8.3 Rilascio . . . . . 56
8.4 Distruzione . . . . . 57

**9 Conclusioni 59**
9.1 Risultati . . . . . 59
9.2 Resoconto . . . . . 60

**Glossario 61**

**Bibliografia 65**

# Elenco delle figure


1.1 Logo zero12 srl . . . . . 1
1.2 Logo VarGroup Spa . . . . . 2

2.1 Tipologie di servizi cloud . . . . . 6
2.2 Logo AWS . . . . . 7
2.3 Distribuzione AWS globale . . . . . 7
2.4 Regions e Availability zones . . . . . 8
2.5 Console AWS . . . . . 9
2.6 Esempio CloudFormation . . . . . 10

3.1 Amazon IAM . . . . . 12
3.2 Esempio di ruolo IAM . . . . . 13
3.3 Esempio di package.json . . . . . 22
3.4 Logo TypeScript . . . . . 22
3.5 Logo VS Code . . . . . 23

4.1 Concetti chiave . . . . . 25
4.2 Esempio di directory di progetto . . . . . 29
4.3 Esempio di cdk.json . . . . . 30

5.1 Diagramma infrastruttura con servizio Pipeline . . . . . 31
5.2 Ridimensionamento di Amazon EC2 Auto Scaling . . . . . 34
5.3 Funzionamento di Amazon Application Load Balancer . . . . . 35
5.4 Esempio di CodePipeline nella console AWS . . . . . 38

6.1 Diagramma infrastruttura con servizio Serverless . . . . . 39
6.2 Funzionamento di Amazon CloudFront . . . . . 41
6.3 Funzionamento di Amazon Cognito . . . . . 42
6.4 Esempio di API Gateway nella console AWS . . . . . 44

7.1 Diagramma infrastruttura con servizio Container . . . . . 47
7.2 Architettura di Kubernetes . . . . . 50
7.3 Amazon EKS . . . . . 51

7.4 Amazon Aurora . . . . . 54

8.1 Comandi cdk per la distribuzione . . . . . 55
8.2 Esempio di stack distribuito in CloudFormation nella console AWS . . . 57

# Capitolo 1

# Introduzione

## 1.1 Azienda

**Figura 1.1:** Logo zero12 srl

Zero12 S.r.l.(figura 1.1) è un'azienda localizzata in due sedi sul territorio italiano: una situata a Padova, presso la quale ho svolto lo stage, e una ad Empoli.
Zero12 si occupa di innovazione e diffusione della cultura digitale in ambito business, e fa della metodologia Agile la sua filosofia portante.
I servizi principali di cui si occupa sono:

- Supporto nel percorso di cloud adoption attraverso la migrazione e/o design di infrastrutture cloud
- Sviluppo di soluzioni software cloud native web e mobile totalmente personalizzate
- Progettazione e sviluppo di soluzioni di machine learning

L'esperienza svolta presso Zero12 si è rivelata positiva e mi ha spinto a voler approfondire le tematiche trattate durante il tirocinio. L'inserimento in azienda, in quanto stagista, è stato facilitato fin dai primi giorni da una stretta collaborazione e disponibilità da parte dei colleghi nell'aiuto alla risoluzione di problemi relativi al progetto, dall'efficiente capacità di comunicazione in determinati ambiti aziendali e produttivi e dal supporto fornito. Il clima in ufficio è professionale e al contempo rilassato, favorito da momenti di confronto e collaborazione, lavori individuali e stand up meeting e scandito da momenti di pause di conversazione o di gioco.

Zero12 è una dei partner AWS e si avvale degli strumenti che essa offre. Amazon Web Services Inc. è un'azienda di proprietà del gruppo Amazon e fornisce servizi per il **cloud computing** sull'omonima piattaforma. Essa distribuisce i propri servizi su scala globale sparsi in 26 regioni geografiche, numero tutt'oggi in crescita, offrendo più di 200 prodotti come networking, database, storage, machine learning e molto altro. Tra questi troviamo ad esempio Amazon Elastic Compute Cloud (EC2) , Amazon Simple Storage Service (S3), Identity and Access Management (IAM) o Amazon Relational Database Service (RDS) . Garantisce tali soluzioni on demand con caratteristiche di scalabilità, high availability e sicurezza, con modalità pay-as-you-go, ossia calcolando il costo basandosi sulle risorse e il loro relativo utilizzo.
Il team di Zero12 è pertanto formato grazie alle certificazioni ufficiali AWS tramite corsi di formazione o aggiornamento, e che il sottoscritto ha seguito per lo studio dei servizi necessari allo svolgimento del progetto di stage previsto.

**Figura 1.2:** Logo VarGroup Spa

Zero12 s.r.l fa parte di Var Group (figura 1.2), società italiana che vanta 23 sedi in Italia e 10 all'estero, facendo di sé una dei principali partner per l'innovazione del settore ICT. Sostiene la competitività delle imprese grazie al Made in Italy 4.0, fornendo supporto alla Digital Transformation e offrendo il proprio servizio in svariati settori come manufacturing, food & wine, meccanica industriale, fashion, furniture, retail & gdo. Var Group appartiene al Gruppo Sesa S.p.A., operatore di riferimento in Italia nell'offerta di soluzioni IT a valore aggiunto per il segmento business, quotata sul segmento STAR del mercato MTA della Borsa Italiana.

## 1.2 Progetto

Il progetto proposto dall'azienda Zero12 srl per lo svolgimento dello stage, intitolato Infrastructure as a code, what else ruota attorno al paradigma dell'Infastructure as Code, anche conosciuto in ambito informatico con l'acronimo di IaC.
Con il termine **infrastruttura**, in questo contesto specifico, si vanno ad intendere tutte quelle risorse, quali ad esempio macchine di calcolo (server), database, reti private o bilanciatori di rete, che vengono impiegate per la distribuzione di uno o più determinati prodotti software, fungendo quindi da fondamenta o impalcatura, sulla quale appoggiarsi. Il termine "Infrastructure as Code" indica l'approccio di gestione e provisioning delle infrastrutture menzionate tramite codice, anziché tramite processi manuali e ripetitivi. Ciò significa che la gestione passa dal livello fisico, ossia hardware, ai grandi data center, grazie al paradigma del **cloud computing**.
L'intera infrastruttura viene pertanto gestita tramite opportuni file che ne contengono le specifiche e ne semplificano la modifica e la distribuzione, codificati utilizzando un linguaggio di alto livello. Questa pratica garantisce riproducibilità, poiché permette il provisioning dello stesso ambiente più volte dalla medesima configurazione.

Un altro aspetto fondamentale delle IaC riguarda la cosiddetta "Immutable infrastructure", termine che va ad indicare l'approccio secondo il quale la gestione ed il deploy di un nuovo ambiente viene effettuato rimpiazzando quello vecchio, piuttosto che aggiornandolo. Ciò si contrappone alla cosiddetta ≪Living infrastructure≫, che si limita a mantenere la stessa istanza effettuando opportuni update, che vanno ad accumularsi nel tempo. La prima soluzione è di gran lunga preferibile ed è quella che viene oggigiorno adottata in quanto riduce la deriva della configurazione ( configuration drift ), ossia il rischio che determinate modifiche lascino un divario tra le configurazioni, garantisce una minore complessità ed evita potenziali guasti comportando una risoluzione dei problemi più semplice.
L'approccio **dichiarativo** è quello che viene generalmente impiegato. Ossia, per effettuare un cambiamento ad una determinata infrastruttura, occorre modificare la dichiarazione delle risorse nei file di configurazione e, a quel punto, sarà poi un opportuno strumento ad occuparsi di effettuare il provisioning gestendo le dipendenze tra esse.

Come si evince, i vantaggi delle Infrastructure as Code sono varie: aiutano ad alleviare il carico sugli sviluppatori incentivando le pratiche di **CI/CD** grazie al controllo di versione del codice, a ridurre fortemente l'insorgere di potenziali errori di configurazione accelerando la risoluzione dei problemi e facilitando i deployments, garantendo una infrastruttura coerente e riproducibile a partire dalle stesse configurazioni.

L'obiettivo dello stage risiede nell'ambito appena descritto. Durante tutto il tiroci-

nio ho studiato i principali servizi AWS (Amazon Web Services) per la progettazione, lo sviluppo e infine il rilascio di determinate infrastrutture in ambiti **pipeline**, **serverless** e **container**, che verranno discussi approfonditamente nelle sezioni dedicate. In particolare il mio compito è stato quello di sviluppare tali infrastrutture in forma di template, ossia definendo uno scheletro di esse e ottenendo i valori della configurazione tramite un file opportuno.

# Capitolo 2

# Introduzione ad AWS

## 2.1 Cloud computing

Quando si parla di cloud computing si va ad intendere la distribuzione on-demand (su richiesta) delle risorse IT tramite Internet, con una tariffazione basata sul consumo. Per molte aziende e organizzazioni di vario tipo, dimensione o settore di interesse, piuttosto che possedere e mantenere i data center e i server fisici, risulta maggiormente conveniente rendere disponibili agli utenti i servizi tecnologici, quali macchine di calcolo, storage e database, tramite un **fornitore cloud**, come avviene per Amazon Web Services.
Per esempio, il cloud è impiegato dalle aziende per i servizi finanziari come base dei propri sistemi di rilevamento di attività fraudolente oppure i realizzatori di videogame utilizzano il cloud per sviluppare videogiochi online.
Si tratta pertanto non di un luogo fisico, ma piuttosto un metodo di gestione delle risorse IT. Il cloud computing presenta una serie di applicazioni e vantaggi:

- **Agilità**: perché permette di accedere in modo semplice a diverse tecnologie, così da poter innovare più rapidamente. Oltre allo sviluppo in tempo reale e allo storage di grandi quantità di dati, è possibile anche distribuire servizi tecnologici con una velocità maggiore rispetto al passato;

- **Scalabilità**: in quanto permette di gestire la quantità di risorse a seconda delle necessità, scalando in verticale e in orizzontale.

- **Risparmio**: poiché permette di evitare enormi spese pagando solo per le risorse realmente utilizzate.

- **High availability**: permette di raggiungere regioni geografiche in pochi minuti, facendo in modo che le applicazioni siano vicino agli utenti finali, riducendo la latenza e migliorandone l'esperienza.

I servizi cloud sono infrastrutture, piattaforme o software ospitati da fornitori di terze parti e resi disponibili agli utenti tramite Internet. Esistono tre tipologie principali di servizi di cloud computing come mostrati in figura 2.1:

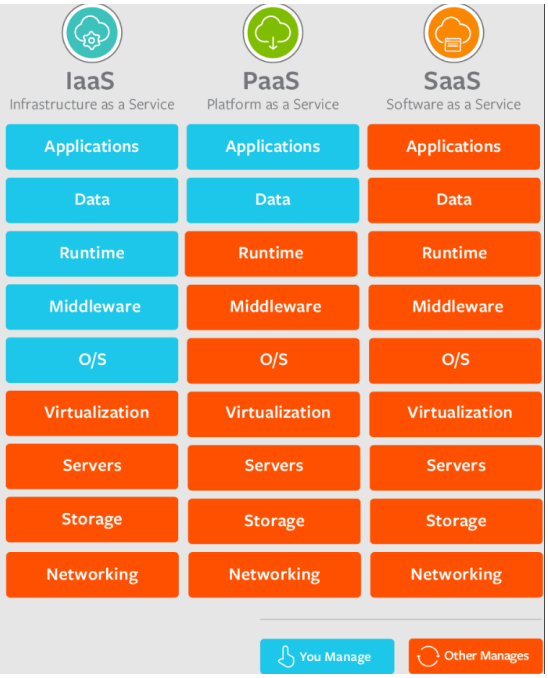


**Figura 2.1:** Tipologie di servizi cloud

- **Infrastructure-as-a-Service** (IaaS): in questo caso un provider di servizi cloud gestisce l'infrastruttura (server, network, archiviazione ecc) tramite una connessione Internet. Dall'altra parte l'utente noleggia l'infrastruttura, a cui vi ha accesso tramite API o un dashboard, e gestisce le risorse come il sistema operativo, le app, i propri dati.

- **Platforms-as-a-Service** (PaaS): in questo caso l'hardware e una piattaforma software applicativa sono forniti e gestiti da un fornitore di servizi cloud esterno, ma l'utente gestisce le app in esecuzione sulla piattaforma e i dati su cui si basa l'app.

- **Software-as-a-Service** (SaaS): in questo caso è un servizio che fornisce un'applicazione software, gestita dal provider di servizi cloud, ai propri utenti. In genere, le app SaaS sono applicazioni Web o app mobili a cui gli utenti possono accedere tramite un browser Web.

## 2.2 Amazon Web Services

**Figura 2.2:** Logo AWS

AWS, acronimo di Amazon Web Services, (figura 2.2) è un piattaforma di proprietà del gruppo Amazon, che offre i propri servizi di cloud computing offrendoli su scala globale.


**Figura 2.3:** Distribuzione AWS globale

Il progetto, lanciato nel 2002, copre al giorno d'oggi gran parte del fatturato di Amazon. Distribuendosi in 26 regioni geografiche (si veda la figura 2.3), numero ancora in crescita, si possono accedere a più di 200 prodotti come ad esempio servizi per networking, database, storage, machine learning e molto altro, secondo le esigenze dell'utente.
L'infrastruttura globale di AWS è organizzata in regioni e zone di disponibilità (vedi figura 2.4). Il concetto di regione va inteso come luogo fisico nel mondo in cui AWS clusterizza i data center. Ognuna è costituita da un minimo di tre zone di disponibilità isolate e fisicamente separate all'interno di un'area geografica. In una regione geografica si trovano diverse zone di disponibilità, e ognuna ospita i data center fisici di Amazon.

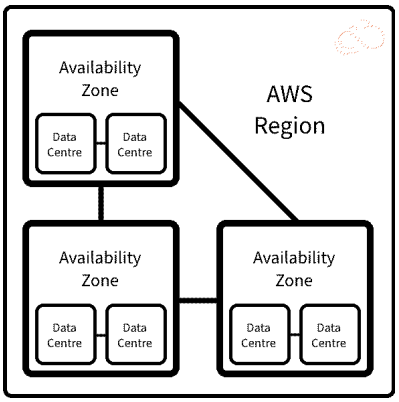


**Figura 2.4:** Regions e Availability zones

Tali zone sono allo stesso tempo geograficamente distanti l'una dall'altra, favorendo l'esecuzione di carichi di lavoro scalabili, tolleranti ai guasti e altamente disponibili. Al seguente link si può avere accesso alla lista globale delle regioni disponibili in AWS.
AWS si fa carico dei compiti più difficili nella distribuzione dei prodotti software, poiché permette di raggiungere i mercati con la sua presenza globale, consentendo di distribuire ed eseguire ogni tipo di applicazione: social network, smart city, videogiochi, streaming video, online banking e molte altre. Grazie al modello di pagamento in base al consumo, si possono scegliere le risorse che meglio rispondono alle proprie esigenze e, in questo modo, le startup accelerano il processo di creazione del prodotto e delle sue funzionalità, mantenendo i costi al minimo. Tutta questa infrastruttura globale è interconnessa tramite una grande rete privata dedicata, altamente disponibile e a bassa latenza, che attraversa oceani e continenti.

## 2.3 Console AWS

La principale via d'accesso per operare tramite AWS è grazie alla AWS Management Console, raggiungibile al seguente link.
La console di gestione fornisce un'interfaccia Web intuitiva per Amazon Web Services, per la quale è possibile effettuare l'accesso utilizzando le credenziali del proprio account.

In Zero12, per permettermi di studiare i servizi, operare direttamente con le risorse generate durante la fase di codifica e quindi verificare gli ambienti sviluppati, mi è stato fornito un account con nome utente e password per l'accesso, con permessi di amministratore in modo tale da non trovare ostacoli, poter sfruttare al massimo la console e apprendere senza vincoli su determinate componenti. In particolare, per evitare costi inutili, sono state utilizzate negli ambienti sviluppati risorse a basso costo, senza andare a togliere allo scopo propedeutico, in quanto facilmente rimpiazzabili con risorse da messa in produzione.

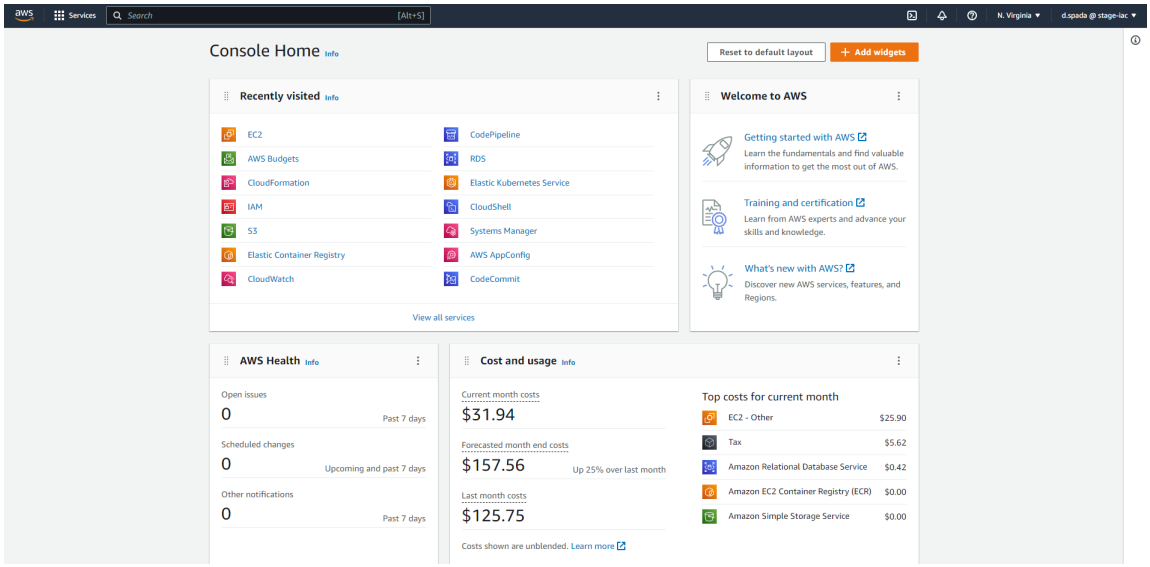


**Figura 2.5:** Console AWS

Al primo accesso, viene visualizzata la home page della console come in figura 2.5, che fornisce una panoramica del proprio account e offre un unico luogo per accedere alle informazioni necessarie per eseguire le attività relative ad AWS, oltre a informazioni sul proprio account e sulla fatturazione. Si può pertanto accedere a ciascuna console di servizio, che apre la rispettiva console in una pagina Web dedicata, fornendo all'utente un'ampia gamma di strumenti per operare sulle rispettive risorse.

## 2.4 AWS CloudFormation

AWS CloudFormation è un servizio che aiuta a modellare e configurare le risorse AWS così da ottimizzare le tempistiche e concentrarsi sulle applicazioni eseguite in AWS. In particolare CloudFormation lavora con modelli e stack.
Un modello CloudFormation è un file di testo in formato JSON o YAML che viene impiegato come template per creare e definire le risorse AWS. Ad esempio, in un modello è possibile descrivere un'istanza Amazon EC2 (macchina di calcolo) tramite le proprietà che possiede.
Tuttavia, i modelli CloudFormation dispongono di funzionalità aggiuntive che permettono di creare gruppi complessi di risorse e riutilizzare tali modelli in più contesti. Un esempio molto semplice è descritto dall'immagine 2.6.

```
{
  "AWSTemplateFormatVersion" : "2010-09-09",
  "Description" : "A simple EC2 instance",
  "Resources" : {
    "MyEC2Instance" : {
      "Type" : "AWS::EC2::Instance",
      "Properties" : {
        "ImageId" : "ami-0ff8a91507f77f867",
        "InstanceType" : "t2.micro"
      }
    }
  }
}
```

**Figura 2.6:** Esempio CloudFormation

Quando si utilizza CloudFormation, si gestiscono risorse correlate, che possono essere create, aggiornate ed eliminate come una singola unità, chiamata stack. Ad esempio, uno stack può includere tutto il necessario per l'esecuzione di un'applicazione Web, ad esempio un server Web, un database e le regole di network.
Tutte le risorse di uno stack sono definite dal modello CloudFormation dello stack. È possibile creare un modello con la descrizione di tutte le risorse che si desiderano (ad esempio, macchine di calcolo EC2 o istanze database RDS) e CloudFormation si occuperà di allestire e configurare tali risorse. Non è necessario crearle e configurarle singolarmente, determinandone le dipendenze, perché è CloudFormation a gestire tutti questi aspetti.

Inoltre, per quelle applicazioni che richiedono la presenza in più regioni, sia per motivi di disponibilità che per ragioni di alta affidabilità in caso di guasti, esso permette di riutilizzare e replicare lo stesso modello creato, generando le risorse in maniera ottimizzata e ripetibile.

# Capitolo 3

# Servizi e Tecnologie fondamentali per lo sviluppo AWS

Nella seguente sezione verranno introdotti ed esposti gli strumenti e le tecnologie studiate durante lo stage, impiegati poi per lo svolgimento dei progetti e delle relative infrastrutture, che permettono di applicare il paradigma precedentemente introdotto dell' Infrastructure as Code. I servizi Amazon di maggiore rilevanza che verranno introdotti e che sono impiegati nelle più comuni infrastrutture sono ad esempio IAM (*Identity and Access Management*), VPC (*Virtual Private Network*), EC2 (*Amazon Elastic Compute Cloud*), RDS (*Relational Database Service*), S3 (*Amazon Simple Storage Service*).
Verranno successivamente descritte le principali tecnologie e strumenti impiegati nella parte di sviluppo come AWS CLI (*AWS Command Line Interface*), AWS CDK (*AWS Cloud Development Kit* ), Node JS per l'installazione delle dipendenze e la gestione dei pacchetti tramite npm, TypeScript come linguaggio di programmazione e Visual Studio Code come ambiente di sviluppo.

## 3.1 Servizi principali per lo sviluppo di infrastrutture cloud

In questa sezione verranno descritti i principali servizi AWS studiati per la progettazione delle più comuni infrastrutture e il loro successivo sviluppo tramite gli strumenti descritti. I restanti servizi saranno trattati nelle sezioni appropriate.

### 3.1.1 Amazon IAM


**Figura 3.1:** Amazon IAM

AWS Identity and Access Management (IAM) è uno dei servizi fondamentali tra gli Amazon Web Services (figura 3.1). Grazie a IAM, è possibile specificare chi o cosa può accedere alle risorse e ai servizi in AWS, gestire centralmente le autorizzazioni in modo molto dettagliato e analizzare gli accessi per affinare le autorizzazioni in AWS.
IAM offre l'infrastruttura necessaria per gestire l'autenticazione e l'autorizzazione che include i seguenti elementi.
Un *principal* è una persona o un'applicazione che può effettuare una richiesta per un'operazione su una risorsa AWS, cioè un oggetto che esiste all'interno di un servizio, e viene autenticato come utente *root* dell'account AWS o come **entità** IAM per effettuare richieste, anche se è buona pratica, come già detto, non utilizzare l'account root per le operazioni ordinarie.

Esistono due tipologie principali di entità: utenti e ruoli.
Un **utente** IAM è un'entità che rappresenta la persona o l'applicazione utilizzati per interagire con AWS e che viene identificato da un codice arn come ad esempio: arn:aws:iam::*account-ID:user/Davide*.
Per impostazione predefinita, un nuovo utente IAM non ha le autorizzazioni per svolgere alcuna operazione ma alla sua creazione è possibile modificarne i permessi.

Un **ruolo** è invece un'identità IAM con autorizzazioni specifiche ed è simile a un utente, in quanto possiede policy di autorizzazione che determinano ciò che l'identità può e non può fare in AWS. Tuttavia, invece di essere associato in modo univoco a una persona, un ruolo è destinato ad essere assunta da chiunque ne abbia bisogno, ad esempio un altro utente nello stesso account o al di fuori di esso o una serie di servizi AWS che devono avere permessi specifici e restrittivi. Inoltre, un ruolo non dispone di creden-

ziali standard ma vengono fornite credenziali di sicurezza temporanee per la sessione del ruolo.

È possibile gestire l'accesso in AWS mediante la creazione di **policy** e il relativo collegamento a identità IAM (utenti, gruppi di utenti o ruoli) o risorse AWS. Una policy è un oggetto in AWS, archiviata come documento JSON che, quando associato a un'identità o a una risorsa, ne definisce le autorizzazioni specifiche.
Come esempio è riportata l'immagine 3.2 di un ruolo nella console AWS nominato "CodeDeployRole" legato ad un servizio (AWS CodeDeploy) e che contiene al suo interno una policy con le relative autorizzazioni.

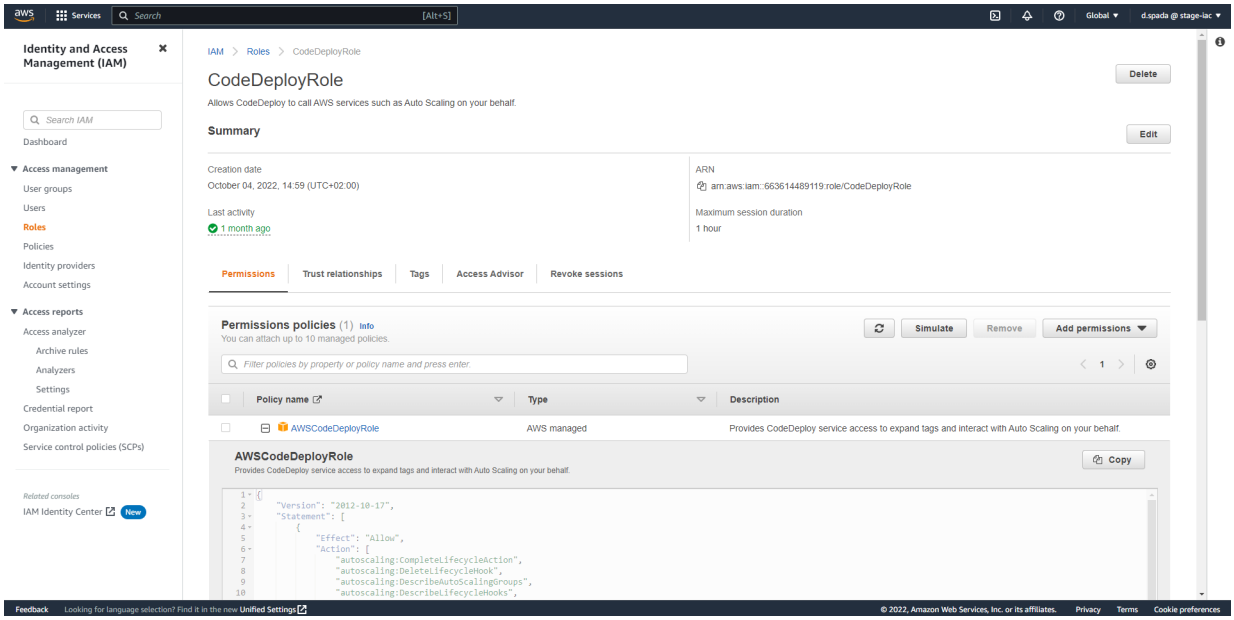


**Figura 3.2:** Esempio di ruolo IAM

Quando un'entità cerca di effettuare una richiesta ad AWS questa richiede alcune informazioni come le azioni o operazioni che intende svolgere, l'oggetto risorsa su cui vengono applicate, il principal che sta agendo e alcuni dati aggiuntivi che forniscono informazioni sulla risorsa. AWS raccoglie le informazioni per valutare e autorizzare la richiesta, valuta queste policy e determina l'approvazione o il rifiuto della richiesta.
Un principal deve essere **autenticato**, ossia aver effettuato l'accesso, utilizzando le proprie credenziali per inviare una richiesta ad AWS. Deve essere inoltre **autorizzato**, cioè deve possedere i permessi appropriati affinché la sua richiesta possa essere accettata ed eseguita affinché possa essere eseguita sulle risorse correlate all'interno dell'account. Queste sono oggetti esistenti all'interno di un servizio come ad esempio un'istanza di una macchina di calcolo ( AWS EC2) , un utente IAM e un bucket di storage AWS S3. Dopo che la richiesta è stata autenticata e autorizzata, AWS approva le azioni o le

operazioni nella richiesta, come ad esempio visualizzare, creare, modificare ed eliminare quella risorsa.

### 3.1.2 Amazon VPC

Amazon Virtual Private Cloud (Amazon VPC) è un servizio che permette di avviare risorse AWS in una **rete privata virtuale** dedicata per il proprio account AWS. È possibile definire più VPC poichè ognuna è isolata logicamente dalle altre. Questa rete virtuale è simile a una comune rete da gestire all'interno del proprio data center, ma con i vantaggi dell'infrastruttura scalabile di AWS. Per una VPC si può, ad esempio, specificare un intervallo di indirizzi IP, aggiungere sottoreti e associare gruppi di sicurezza.

#### Assegnazione di indirizzi IP

L'assegnazione di indirizzi IP ad un VPC permette alle risorse che vi risiedono di comunicare tra loro e con le risorse su internet.
Quando viene creato un VPC è necessario assegnargli un **blocco CIDR** IPv4 (un intervallo di indirizzi IPv4), un blocco CIDR IPv6 o entrambi. Per semplicità tratteremo solo il primo caso. Viene infatti impiegata la notazione CIDR (Classless Inter-Domain Routing) per rappresentare un indirizzo e la sua maschera di rete.
In particolare, il formato per IPv4 è il seguente:

- un singolo indirizzo è a 32 bit con 4 gruppi composti da un massimo di 3 cifre decimali (da 0 a 255). Ad esempio, 10.0.1.0.
- Un blocco CIDR, che identifica un intervallo di indirizzi singoli, ha 4 gruppi composti da un massimo di 3 cifre decimali (da 0 a 255), seguiti da una barra e un numero compreso tra 0 e 32. Ad esempio, 10.0.0.0/16.

Gli indirizzi IPv4 possono essere privati, ovvero non raggiungibili tramite Internet e si possono utilizzare per la comunicazione tra le istanze presenti nel VPC, oppure pubblici. In quest'ultimo caso un indirizzo IP pubblico è associato ad un indirizzo privato primario tramite conversione degli indirizzi di rete (network address translation, NAT).

#### Sottoreti o Subnets

Una sottorete, o subnet, corrisponde ad un intervallo degli indirizzi IP assegnati al VPC e le istanze delle risorse Amazon vi devono risiedere all'interno. Quando si crea un subnet è necessario specificare il blocco CIDR IPv4 per la sottorete, che è a sua volta un **sottoinsieme del blocco** CIDR del VPC. Inoltre, tutte le sottoreti hanno un attributo che determina se nella sottorete viene assegnato un indirizzo pubblico o

privato.
Ogni sottorete deve risiedere all'interno di una sola zona di disponibilità (availability zone). Avviando le istanze in zone di disponibilità separate, sarà possibile proteggere le applicazioni da potenziali fallimenti di una singola posizione. Le sottoreti sono suddivise tra pubbliche e private:

- Subnet pubblica: il traffico della sottorete viene instradato a Internet tramite un Internet gateway.
- Subnet privata: il traffico della sottorete non può raggiungere Internet tramite un Internet gateway ma richiederebbe un dispositivo NAT per la conversione degli indirizzi.

#### Internet gateway

Un Internet gateway è un componente VPC che consente la comunicazione tra il VPC e Internet. Consente pertanto alle risorse nelle sottoreti pubbliche di connettersi a Internet se la risorsa ha un indirizzo IPv4 pubblico. Allo stesso modo, le risorse su Internet possono avviare una connessione alle risorse nella sottorete.
Per abilitare l'accesso a o da Internet per le istanze in una sottorete in un VPC, è necessario collegare il gateway alla corrispondente VPC, aggiungere un percorso alla route table (tabella di instradamento, spiegato sotto) della sottorete che indirizza il traffico diretto a Internet verso il gateway Internet, assicurandosi che le istanze abbiano un indirizzo IPv4 pubblico e ci siano i corretti permessi di sicurezza.

#### NAT gateway

Un NAT gateway è un servizio NAT (Network Address Translation) che viene utilizzato in modo che le istanze in una sottorete privata possano connettersi a servizi in Internet, ma i servizi esterni non possono avviare una connessione con tali istanze.
Nella pratica, si crea un NAT gateway pubblico in una sottorete pubblica e si indirizza poi il traffico dal NAT all'Internet gateway per il VPC. A questo punto, per le istanze (come macchine di calcolo) nelle subnets private che richiedono l'accesso a Internet, occorre instradare il traffico delle subnet verso il NAT gateway, che si occuperà poi lui del resto.

#### Routing tables

Una Routing table (o tabella di route) contiene un insieme di regole, denominate routes, che determinano dove viene indirizzato il traffico di rete.
Ciascun percorso specifica una destinazione e un target verso il quale è indirizzato. Ad esempio, per consentire ad una sottorete di accedere a Internet, è sufficiente aggiungervi il percorso alla tabella di instradamento. La destinazione del percorso è 0.0.0.0/0, che

rappresenta tutti gli indirizzi IPv4, la destinazione è l'Internet gateway collegato al VPC. Se una sottorete è associata a una tabella di route che dispone di una route a un Internet gateway , è nota come sottorete pubblica. Viceversa, se non dispone di una route a un Internet gateway, è nota come subnet privata.

#### Sicurezza

AWS VPC fornisce inoltre caratteristiche per aumentare e monitorare la sicurezza della rete e delle sottoreti associate. A questo proposito vengono impiegati i seguenti strumenti:

- **Security groups**: consentono o negano traffico specifico in entrata e in uscita a livello di risorsa, come ad esempio ad una macchina di calcolo EC2. Quando viene avviata un'istanza, è possibile associare tale istanza a uno o più gruppi di sicurezza e, se non specificato, viene automaticamente associata al gruppo di sicurezza predefinito per il VPC.
- **Lista di controllo degli accessi** (ACL): permettono o negano traffico in entrata e in uscita specifico a livello di sottorete.
- **Log di flusso**: i log di flusso VPC acquisiscono informazioni sul traffico IP da e verso le interfacce di rete nel VPC.

### 3.1.3 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) fornisce **capacità di calcolo** nel cloud di AWS tramite ambienti di elaborazione virtuale. È possibile istanziare un numero di server EC2 secondo le proprie necessità, configurare aspetti relativi alla sicurezza e i servizi di rete, nonché gestire l'archiviazione.
Grazie alla caratteristica di scalabilità verticale, consente di gestire le variazioni di richiesta riducendo la necessità di elaborare previsioni relative al traffico.
Amazon EC2 offre varie caratteristiche configurabili, tra le più rilevanti troviamo ad esempio:

- **VPC e subnet**: la rete privata e il subnet nel cloud AWS a cui è associata l'istanza.
- **Amazon Machine Image** (AMI): ossia modelli preconfigurati per le istanze contenenti i pacchetti di bit necessari per il server (compresi il sistema operativo e il software aggiuntivo).
- **Instance types**: varie configurazioni di CPU, memoria, storage e capacità di rete per inizializzare l'istanza.

- **Instance store volumes**: volumi di archiviazione per i dati temporanei che vengono eliminati quando l'istanza viene terminata.

- **Security groups**: uno o più gruppi di sicurezza che specificano i protocolli, le porte e gli intervalli IP di origine che l'istanza può raggiungere.

### 3.1.4 Amazon RDS

Amazon Relational Database Service (Amazon RDS) è un servizio che consente di istanziare **database relazionali** nel cloud AWS, semplificandone la configurazione, l'uso e il dimensionamento. Un database relazionale è una raccolta di elementi dati tra i quali sussistono relazioni predefinite. Questi elementi sono organizzati sotto forma di set di tabelle con righe e colonne. È quindi possibile interrogare tale database per estrarne le informazioni e i dati contenuti e questo avviene tramite il linguaggio **SQL**, acronimo di Structured Query Language, ovvero l'interfaccia principale usata per comunicare con i database relazionali. Per la maggior parte delle distribuzioni, RDS è la soluzione maggiormente impiegata quando si ha la necessità di utilizzare un database, spesso abbinato con una macchina di calcolo EC2.

Una istanza database è un ambiente isolato che rappresenta l'elemento di base di Amazon RDS. L'istanza database a sua volta può contenere uno o più database creati dall'utente. RDS, come per EC2, è configurabile sotto vari aspetti e presenta alcune caratteristiche vantaggiose. Di seguito sono elencate le più rilevanti:

- **DB engine**: è possibile specificare il motore specifico per l'istanza. Quelli supportati sono MariaDB, Microsoft SQL Server, MySQL, Oracle e PostgreSQL.

- **DB instance**: si può configurare la capacità di calcolo e di memoria di un'istanza.

- **Multi-AZ**: permette di eseguire l'istanza in diverse zone di disponibilità. Amazon esegue automaticamente il provisioning e la manutenzione di una o più istanze database secondarie in una zona di disponibilità diversa.

- **Backup**: Si possono impostare backup automatici o snapshot di backup, utili per ripristinare un database in maniera efficiente e sicura.

- **Autorizzazioni**: Permette di controllare chi può accedere ai tuoi database RDS tramite Amazon IAM per definire utenti e autorizzazioni.

- **Security groups**: tramite i gruppi di sicurezza, controlla l'accesso consentendolo esclusivamente agli intervalli di indirizzi IP o alle istanze Amazon EC2 specificate.

### 3.1.5 Amazon S3

Amazon Simple Storage Service (Amazon S3) è un servizio di **archiviazione** di oggetti utilizzato per archiviare e proteggere qualsiasi quantità di dati in un'ampia gamma di casi d'uso. Ad esempio per data lake, siti Web, applicazioni mobili, backup, archivi, dispositivi IoT e molto altro. Offre quindi caratteristiche di elevata scalabilità, disponibilità dei dati, sicurezza e prestazioni.
A differenza di RDS, si può accomunare S3 ad un file system, che quindi non impiega un modalità strutturata dei dati tramite tabelle e pertanto non impiega un linguaggio per interrogare il sistema.

In Amazon S3 i dati sono archiviati come **oggetti** nei cosiddetti **bucket**.
Un oggetto è un file di qualsiasi tipo con tutti i suoi metadati, mentre un bucket è un container che li contiene. Questi possono contenere, a detta di Amazon, un numero illimitato di oggetti, e fino a un massimo di 100 bucket per account.
Per caricare un oggetto è innanzitutto necessario creare un bucket S3, assegnarvi un nome univoco e una regione specifica e, per impostazione predefinita, tale bucket sarà privato. A tale oggetto verrà quindi assegnato un chiave che costituirà l'identificatore. Ad esempio, se l'oggetto denominato *photos/sunset.jpg* è archiviato nel bucket *DOC-EXAMPLE-BUCKET* della regione Stati Uniti occidentali (*us-west-1*), è indirizzabile tramite l'URL:
*https://DOC-EXAMPLE-BUCKET.s3.us-west-1.amazonaws.com/photos/sunset.jpg*.
È possibile concedere esplicitamente le autorizzazioni di accesso utilizzando policy di bucket, policy Amazon IAM o liste di controllo accessi (ACL). Le autorizzazioni legate ad un bucket , che utilizzano la sintassi delle policy basata su JSON, si applicano a tutti gli oggetti al suo interno e sono impiegate per aggiungere o negare autorizzazioni per gli oggetti.

## 3.2 Tecnologie per lo sviluppo di infrastrutture AWS

### 3.2.1 AWS CLI

AWS Command Line Interface (AWS CLI) è uno strumento open source che consente di interagire con i servizi AWS utilizzando istruzioni da **riga di comando**. Con una configurazione minima, consente di iniziare ad eseguire i comandi che implementano funzionalità equivalenti a quelle fornite dalla versione browser della console AWS in un programma da prompt dei comandi. Nei sistemi Linux e MacOS utilizza shell comuni come bash, zsh, etcsh, per i sistemi Windows il command prompt o PowerShell, oppure tramite programma terminale remoto come PuTTY, SSH o AWS Systems Manager.
La CLI ha quindi accesso alle **API** pubbliche che i servizi AWS espongono e consente di sviluppare script di shell per gestire le risorse. Oltre ai comandi di basso livello equivalenti alle API, diversi servizi AWS offrono personalizzazioni che possono includere comandi di livello più elevato e che vanno a semplificarne l'utilizzo con un'API complessa.

#### Installazione

Per poter utilizzare AWS CLI è necessario:

1. Creare un account AWS, alla seguente pagina di registrazione. La registrazione genererà un account utente root AWS, che ha accesso a tutte le risorse e servizi.
2. Creare un account utente IAM. Generalmente è buona pratica creare un utente all'interno dello stesso account con permessi d'amministratore e utilizzare l'account root solo per azioni strettamente legate a permessi d'accesso.
3. Installare l'ultima versione della AWS CLI. Al seguente link è possibile seguire i passaggi specifici per ogni sistema operativo supportato.

#### Configurazione

Aprendo un programma terminale è possibile configurare AWS CLI eseguendo il comando *"aws configure"*. Questo porterà alla richiesta di quattro campi dati.

1. AWS Access Key ID
2. AWS Secret Access Key

I campi dati 1 e 2 corrispondono ad una key pair IAM di accesso. Per l'accesso alla CLI è necessario impostare una coppia di chiavi, composte da una chiave ID e una chiave segreta, necessarie per firmare le richieste verso AWS. La CLI utilizza tali chiavi per autenticarsi e quindi operare sui servizi. Per generarle è necessario operare da console dalla pagina del proprio account utente (non root), scaricare localmente un file .csv con le chiavi e impostarle durante la seguente configurazione.

3. Default region name

Il campo 3 (Default region name) indica la Region AWS di default verso la quale inviare le richieste. Generalmente è consigliato impostare la region più vicina alla propria posizione in modo tale da ridurre la latenza.

4. Default region name

Il campo 4 specifica la modalità di formattazione dei risultati. Se non si specifica un formato di output, JSON viene utilizzato per impostazione predefinita.
Per maggiori informazioni sugli step da seguire consultare il link seguente.
Successivamente AWS CLI archivia tali dati in un due file, raggiungibili nella cartella .aws della home directory locale, che suddividono le informazioni per profili. Per impostazione predefinita, AWS CLI utilizza le impostazioni presenti nel profilo denominato *default*. In *~/.aws/credentials* si trovano le credenziali del profilo generato, denominato *default*. In *~/.aws/config* si trovano le informazioni restanti, configurabili opzionalmente con altri parametri.

### 3.2.2 AWS CDK

#### Introduzione

AWS CDK, acronimo di Cloud Development Kit, è un framework per la definizione di infrastrutture cloud tramite codice, affidabili, scalabili ed efficienti tramite l'alta espressività fornita dai linguaggi di programmazione come TypeScript, JavaScript, Python o Java.
I vantaggi che derivano nell'impiegare questo approccio sono molteplici, in quanto racchiude in sé i concetti del paradigma ***Infrastructure as Code***.

Permette di sfruttare la massima espressività del codice creando **costrutti di alto livello**, sfruttando idiomi di programmazione come condizionali, cicli, ereditarietà o composizione per modellare la progettazione del sistema e per definire infrastrutture completamente personalizzabili secondo necessità. Inoltre impiega pratiche di ingegneria del software come i test di unità o il controllo del codice sorgente, rendendo l'applicazione robusta e sempre pronta al rilascio (deploy) . Sfrutta in particolare la potenza offerta da Cloudformation, infatti l'output di un programma CDK corrisponde ad un modello AWS Coudformation, a seguito di un processo di sintesi. Viene sfruttato importando i modelli esistenti per la definizione delle risorse e successivamente per eseguire in fase di rilascio la distribuzione dell'infrastruttura, in modo ripetibile e prevedibile, con rollback in caso di errori.

#### Prerequisiti per l'utilizzo di AWS CDK

Per utilizzare il Cloud Development Kit è necessario innanzitutto avere a disposizione un account AWS e le corrispondenti chiavi d'accesso configurate, come precedentemente descritto. Successivamente è necessario installare una versione di Node JS maggiore o uguale alla 10.13, poiché utilizzato dalle applicazioni CDK, indipendentemente dal linguaggio di programmazione scelto per lo sviluppo.
Il linguaggio scelto invece influirà sulla sulla parte di codifica vera e propria poiché andrà specificato durante l'inizializzazione di un progetto in CDK. Verrà inoltre adottato, come ambiente di sviluppo integrato (IDE), Visual Studio Code.

### 3.2.3 Node JS & npm

Node JS è un **runtime JavaScript** open-source e cross-platform costruito sul motore JavaScript V8 di Google Chrome. È un **ambiente di esecuzione** che permette di eseguire codice Javascript come un qualsiasi linguaggio di programmazione. Javascript è nato come linguaggio web lato client e che può essere eseguito unicamente all'interno di un browser. Node JS tuttavia è riuscito a far uscire JavaScript da questa limitazione trasformandolo in linguaggio di programmazione come qualsiasi altro, potendolo utilizzare sia lato client che lato server. Sono infatti presenti moduli per gestire I/O file system, networking, dati binari, crittografia, e molto altro.
La libreria standard di Node JS offre un ampio insieme di funzioni asincrone che gli permettono di eseguire operazioni I/O non bloccanti. Tutto questo gli permette di gestire connessioni concorrenti con un singolo server migliorando la gestione delle risorse, evitando sprechi e aumentando l'efficenza d'esecuzione.

npm, acronimo di **Node Package Manager**, è il gestore di pacchetti ufficiale che viene installato con la piattaforma Node JS. Si tratta di un'interfaccia a riga di comando che aiuta nell'installazione dei pacchetti, nella gestione delle versioni e nella gestione delle dipendenze. npm fa anche riferimento al registro, un grande database pubblico di applicazioni JavaScript, e al sito web che consente di scoprire pacchetti, impostare profili e gestire altri aspetti dell'esperienza npm. In particolare, le dipendenze all'interno di un progetto vengono specificate in un file chiamato package.json e installate localmente in una cartella chiamata *node_modules*. Fra queste possiamo trovare *'aws-cdk-lib'*, contenente i costrutti AWS. Di seguito viene riportato un esempio nell'immagine 3.3 del file package.json:


**Figura 3.3:** Esempio di package.json

Troviamo in particolare:

- scripts: definizione di comandi che possono essere invocati mediante "*npm run <nome script>*"
- devDependencies: indica i pacchetti esterni utilizzati per lo sviluppo dell'applicazione ma non sono strettamente necessari per l'esecuzione della stessa;
- dependencies: indica i pacchetti esterni all'applicazione necessari al suo funzionamento.

È possibile scaricare e installare Node JS dal seguente link.

### 3.2.4 TypeScript


**Figura 3.4:** Logo TypeScript

Il linguaggio di programmazione impiegato per lo sviluppo è TypeScript (immagine del logo 3.4). TypeScript è un **linguaggio open source** ed in continua evoluzione, nato come una estensione di JavaScript e sviluppato da Microsoft. La caratteristica principale rispetto a JavaScript è l'aggiunta dei tipi a variabili, parametri e valori ritornati dalle funzioni, andando a colmare alcune mancanze di quest'ultimo e fornendo pertanto una **maggiore espressività**, senza mancare della compatibilità fra i due.
Normalmente si utilizza per grandi progetti, siano essi front-end come web app o back-end con Node JS. Per poter essere usato nei browser o in ambienti Node JS, il codice

TypeScript ha bisogno di essere compilato e convertito in codice JavaScript, e per questo motivo la libreria ha incorporato un compilatore che funziona con Node JS. Vale anche il contrario, ossia qualunque programma scritto in JavaScript è anche in grado di funzionare con TypeScript senza la necessità di riscrivere tutto. Infatti, per agevolare questa compatibilità tra i due linguaggi, viene utilizzato un *transpiler*, ossia uno strumento di transpilazione che esegue una traduzione da un codice sorgente ad un altro, in questo caso specifico da codice sorgente TypeScript a codice sorgente JavaScript.
Queste caratteristiche lo rendono pertanto molto versatile e ne motivano la costante crescita di popolarità riscontrata fin dalla sua nascita.
In Zero12, infatti, per la codifica delle infrastrutture e quindi per la parte di sviluppo del progetto di stage, viene impiegato come linguaggio di programmazione, poiché è stato il primo linguaggio supportato per lo sviluppo in cdk, vista anche la maggiore quantità di esempi e tutorial all'interno della documentazione ufficiale di AWS e nelle guide online.

### 3.2.5 Visual Studio Code

Visual Studio Code (immagine del logo 3.5) è un **editor di codice sorgente** sviluppato da Microsoft ed è supportato per Windows, Linux e macOS. È un software libero e gratuito, anche se la versione ufficiale è sotto una licenza proprietaria.
Inoltre poggia il suo funzionamento su Electron, noto framework con cui è possibile realizzare applicazioni Node.js, veloce e leggero, non soffrendo delle problematiche in termini di performance che affliggono alcune app JavaScript.

**Figura 3.5:** Logo VS Code

Visual Studio Code permette in particolare di evidenziare la sintassi di ciascun linguaggio di programmazione, integrare il supporto per il debugging, effettuare il controllo Git sul codice, aiutare nella fase di codifica tramite il completamento automatico delle istruzioni, fornendo inoltre la possibilità di mantenere aperti più file affiancandone il contenuto in più schede e molto altro ancora.
Il vero punto di forza di Visual Studio Code sono le estensioni, grazie alle quali è possibile ampliare notevolmente le funzionalità del programma.

# Capitolo 4

# Progettazione e Sviluppo di infrastrutture AWS

## 4.1 Concetti chiave

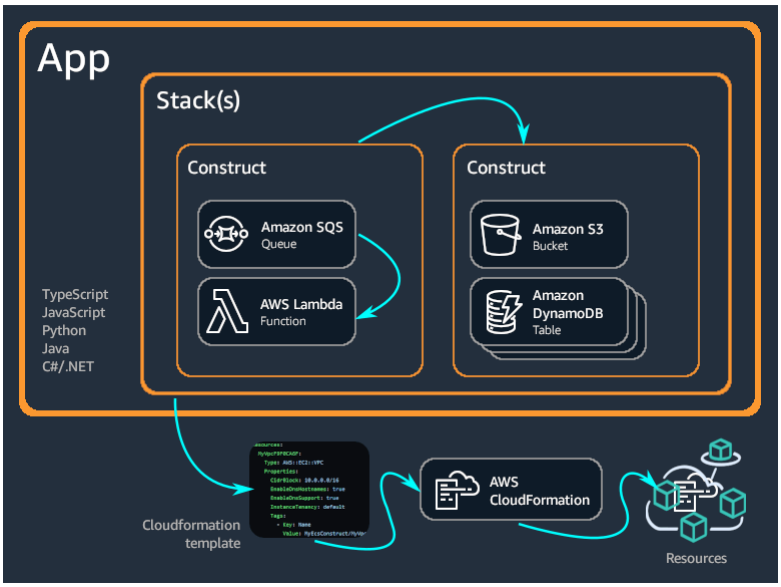


**Figura 4.1:** Concetti chiave

Un'applicazione AWS CDK viene scritta tramite uno dei linguaggi precedentemente menzionati, per definire l'infrastruttura AWS. D'ora in poi, negli esempi e snippet di codice verrà utilizzato TypeScript.
Un'**App** agisce da container per ospitare al suo interno uno o più Stack. Gli **Stack** a loro volta contengono **Costrutti**. I costrutti definiscono le risorse AWS, componibili tra loro per costruire astrazioni di livello superiore. Si veda in riferimento la figura 4.1

### Construct

L'AWS CDK include una libreria di costrutti chiamata AWS Construct Library, organizzata in vari moduli e che contiene i vari costrutti per ogni servizio, incluse le classi base come Stack e App che vengono utilizzate nella maggior parte delle applicazioni CDK.
I costrutti sono gli elementi costitutivi di base delle app e rappresentano un componente cloud e incapsulano tutto il necessario affinché AWS CloudFormation possa creare il componente. Un costrutto può rappresentare una singola **risorsa** AWS ma può anche rappresentare un'astrazione costituita da più risorse correlate. Alcune risorse sono ad esempio istanze di Amazon S3 o Amazon RDS.
Esistono 3 livelli di costrutti:

- L1: sono costrutti di basso livello, che chiamiamo CFN Resources e che rappresentano direttamente tutte le risorse disponibili in AWS CloudFormation. Necessitano di configurare in modo esplicito tutte le proprietà delle risorse
- L2: sono costrutti che forniscono un livello di astrazione superiore. Forniscono funzionalità simili, ma incorporano le impostazioni predefinite e la logica di base che altrimenti richiederebbe un costrutto L1.
- L3: per completezza, anche se non impiegati nel progetto di stage, questi costrutti sono conosciuti anche come **pattern** e nella maggior parte dei casi coinvolgono più tipi di risorse.

La **composizione** diventa quindi il modello chiave poiché un costrutto di alto livello può essere composto da qualsiasi numero di costrutti di livello inferiore.

Tutti i costrutti accettano tre parametri quando vengono inizializzati:

- **scope**: Il genitore del costrutto, che può essere uno stack o un altro costrutto. Di solito this, che rappresenta l'oggetto corrente, per l'ambito.
- **id**: un identificatore che deve essere univoco all'interno di questo ambito. Viene utilizzato per generare identificatori univoci in Cloudformation.
- **props**: un insieme di proprietà o argomenti di parole chiave che definiscono la configurazione iniziale a seconda del costrutto.

Vediamo un esempio di costrutto L2, maggiormente impiegato per la parte di codifica, che istanzia un bucket Amazon S3 creando un'istanza della classe *Bucket*.

```
import * as s3 from 'aws-cdk-lib/aws-s3';
// "this" is HelloCdkStack
const myBucket = new s3.Bucket(this, 'MyFirstBucket', {
  versioned: true
});
```

**Listing 4.1:** Esempio di Costrutto

Oltre a utilizzare costrutti esistenti, è possibile estendere la classe Construct e tramite la composizione definire più risorse al suo interno, per poter essere riutilizzato in altre parti dell'app.
Dopo aver istanziato un costrutto, l'oggetto espone una serie di metodi e proprietà che consentono di interagire con esso e passarlo come riferimento ad altre parti. L'esempio seguente concede al gruppo IAM *data-science* l'autorizzazione a leggere dal bucket S3 *raw-data* tramite il metodo *grantRead()*.

```
const rawData = new s3.Bucket(this, 'raw-data');
const dataScience = new iam.Group(this, 'data-science');
rawData.grantRead(dataScience);
```

**Listing 4.2:** Esempio di autorizzazione IAM

### App e Stack

Per eseguire il provisioning, tutti i costrutti che rappresentano le risorse AWS devono essere definiti nell'ambito di uno stack. Un'app, essendo un contenitore per uno o più stack, funge da ambito di ogni stack. AWS CDK deduce le dipendenze tra gli stack in modo che possano essere distribuiti nell'ordine corretto. L'esempio seguente dichiara una classe stack denominata *MyFirstStack* che include un singolo bucket Amazon S3.

```
class MyFirstStack extends Stack {
    constructor(scope: Construct, id: string, props?: StackProps) {
      super(scope, id, props);

      new s3.Bucket(this, 'MyFirstBucket');
    }
  }
```

**Listing 4.3:** Esempio di Stack

Tuttavia questo codice ha dichiarato solo uno stack. Affinché venga sintetizzato in un modello AWS CloudFormation e distribuito, deve essere istanziato nel contesto di App, come mostrato di seguito.

```
1 const app = new App();
2 new MyFirstStack(app, 'hello-cdk');
```

**Listing 4.4:** Esempio di App

#### Environments

Ogni stack nell'app è associata a un ambiente (env). Un ambiente è una coppia di valori costituita dall'account AWS di destinazione e la regione in cui lo stack deve essere distribuito. La regione viene specificata utilizzando un codice. Per un elenco delle regioni si veda il link seguente. L'esempio seguente specifica l'*env* nel quale verrà deployato *MyFirstStack*. Se non viene specificato, CDK effettuerà il deploy nel profilo di default configurato precedentemente.

```
1 const envUSA = { account: '8373873873', region: 'us-west-2' };
2 new MyFirstStack(app, 'first-stack-us', { env: envUSA });
```

**Listing 4.5:** Esempio di environment

## 4.2 Inizializzazione di un'app in CDK

Dopo la configurazione di AWS CLI e l'installazione di Node JS, è necessario installare AWS CDK Toolkit da riga di comando tramite *npm install -g aws-cdk*, che andrà a scaricare l'ultima versione disponibile.
L'AWS CDK Toolkit permette di utilizzare lo strumento 'cdk' che, da riga di comando, consente di interagire con l'applicazione AWS CDK, ossia di svolgere varie operazioni come ad esempio inizializzare un progetto, produrre il modello di AWS Cloudformation, interrogare tale modello o effettuare il deploy.

Ogni app deve trovarsi all'interno di una propria directory (figura 4.2) con i moduli delle proprie dipendenze. A questo punto è possibile inizializzare un progetto aprendo il terminale in tale cartella e inviare il comando *cdk init app –language typescript* che , tramite il flag *–language* andrà a configurare come linguaggio TypeScript di default, in questo caso specifico. Aprendo la cartella in VS Code si può notare come il comando abbia inizializzato una serie di file e cartelle all'interno della directory per aiutare a organizzare il codice sorgente per l'app AWS CDK.

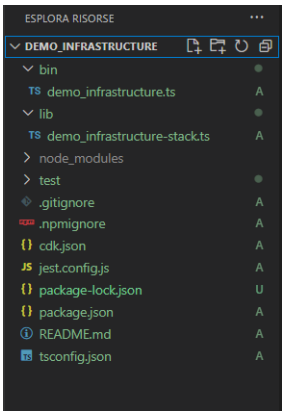


**Figura 4.2:** Esempio di directory di progetto

Analizziamo le risorse più rilevanti nella cartella:

- la cartella *lib* contiene le definizioni degli stack principali che quindi contengono, come descritto, le varie risorse ( o costrutti) AWS.
- la cartella *bin* è il punto di ingresso dell'applicazione CDK andando a caricare gli stack definiti in *lib*.
- *node_modules* è una cartella, come precedentemente descritto, gestita da npm che include tutte le dipendenze del progetto e viene automaticamente creata dal comando *cdk init*.
- *package.json* è il manifesto del modulo npm. Include informazioni come il nome dell'app, la versione, le dipendenze e script di build.
- *cdk.json* dice al toolkit come eseguire l'app. Relativo all'esempio sarà: "*npx ts-node –prefer-ts-exts bin/demo_infrastructure.ts*". Funge inoltre da file di configurazione per il template dell'infrastruttura, come spiegato di seguito.

## 4.3 Configurazione del template di progetto

Arriviamo ora al cuore dell'obiettivo del progetto di stage. Tutte le infrastrutture sviluppate, che verranno descritte in seguito, avranno la caratteristica comune di essere in forma "**templatizzata**". Ciò significa che i parametri, e in particolare le proprietà di ogni risorsa istanziata, non vengono inseriti in 'chiaro' nel codice, bensì, sono ottenuti tramite un metodo opportuno da un file di configurazione.
Questo permette di definire infrastrutture standardizzate con uno scheletro comune, ma che in base ai valori nel file di configurazione possono assumere proprietà differenti. Per fare un esempio, un'istanza di una macchina di calcolo EC2 richiede tra le sue proprietà la vpc a cui fa riferimento, per poter essere inizializzata. Il valore non è scritto direttamente ma viene passato da un file esterno.

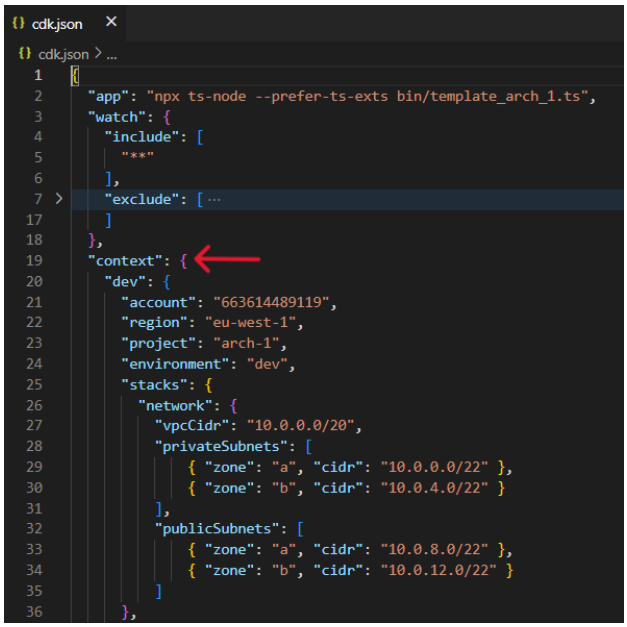


**Figura 4.3:** Esempio di cdk.json

Viene infatti impiegato il **context** ( o contesto ), che contiene elementi che sono coppie chiave-valore e che possono essere associate a un'app, uno stack o un costrutto, e vengono forniti dal file *cdk.json* come mostrato nell'esempio in figura 4.3. Le chiavi di contesto sono stringhe mentre i valori possono essere di qualsiasi tipo supportato da JSON: numeri, stringhe, matrici o oggetti. I valori di contesto sono forniti tramite la funzione *getConfig(app: App)*. Specificando la chiave *"context"* nel file *cdk.json*, il metodo prende, tramite l'opzione *–context* (*-c* in breve) del comando CDK, le informazioni contenute, come si può vedere dalla figura. Si può notare infatti come nel contesto si possano inserire i valori in maniera strutturata secondo la sintassi JSON, ad esempio riguardo l'account, la regione, gli indirizzi IP della parte di network del VPC e quant'altro.
Le funzioni che permettono di passare le proprietà dal cdk.json ai costrutti in fase di rilascio sono contenute nella cartella *common_config*, all'interno di ogni progetto.

### API Reference

Per lo sviluppo delle infrastrutture ho fatto riferimento all'*API reference*, che contiene informazioni su AWS Construct Library e altre API fornite da AWS CDK. È organizzato in moduli e sottomoduli e per ogni servizio è messo a disposizione un overview su come utilizzare le API.
In particolare, nella fase di codifica, ho fatto riferimento ad AWS CDK v2, in quanto il vecchio CDK v1 è entrato in manutenzione e terminerà l'1 giugno 2023. Al seguente link si può raggiungere l'API reference.

# Capitolo 5

# Template di infrastruttura con servizio Pipeline

Nella seguente sezione esporrò il primo progetto di infrastruttura in forma templatizzata. Il diagramma in figura 5.1 mostra le risorse AWS all'interno di AWS Cloud, ossia lo spazio all'interno dell'account AWS, e le relazioni, indicate dalle frecce, che sussistono fra loro. Il progetto è disponibile in GitHub nel repository pubblico raggiungibile a questo link.

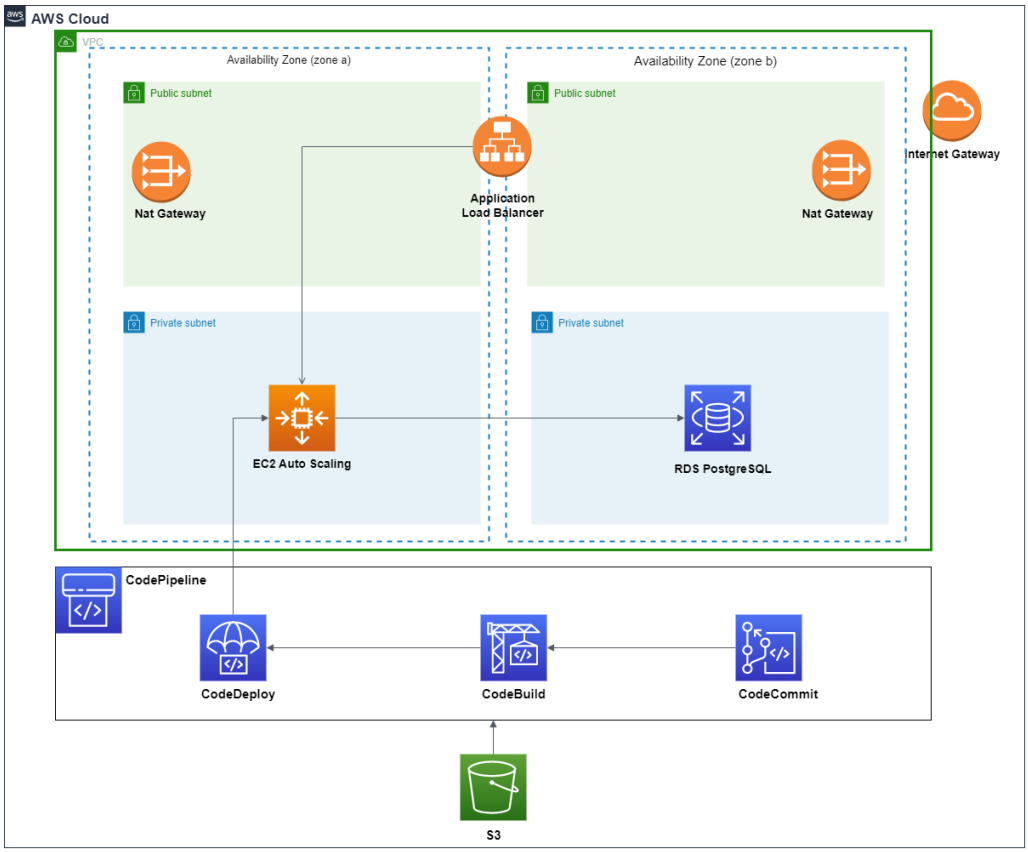


**Figura 5.1:** Diagramma infrastruttura con servizio Pipeline

I principali servizi che la compongono sono:

- **Amazon VPC**: servizio che definisce la componente di network e altri servizi funzionali ad esse come NAT Gateway e Internet Gateway.
- **Amazon EC2**: Auto Scaling: servizio che riprende i concetti di Amazon EC2 fornendo proprietà di scalabilità.
- **Amazon RDS**: servizio di database relazionale con motore MySQL.
- **Amazon Application Load Balancer**: servizio per la gestione del traffico di rete
- **Amazon CodePipeline**: servizio per creare una pipeline di CI/CD
- **Amazon S3**: è servizio di storage

Di seguito fornirò un'analisi dei componenti dell'infrastruttura e della sua logica, soffermandomi sui servizi non introdotti in precedenza.

## 5.1 Amazon VPC

Il VPC definisce l'**infrastruttura di rete**, o network, che permette la comunicazione tra le risorse e internet.
La prassi comune consiste nell'istanziare, all'interno di un vpc, almeno **2 sottoreti per tipologia**, cioè almeno 2 subnet pubbliche e 2 subnet private. Queste, per il progetto, sono state inserite nella region *eu-west-1*, che corrisponde ai data center presenti in Irlanda. A loro volta le sottoreti del vpc sono state distribuite in due zone di disponibilità diverse (zona A e zona B), in modo tale che ogni zona abbia due sottoreti , cioè una pubblica e una privata, come da figura.
La motivazione risiede nel fatto che, se una zona di disponibilità dovesse fallire, e quindi i rispettivi data center fisici non fossero disponibili temporaneamente, l'altra zona di disponibilità assicurerebbe comunque la fruibilità del servizio. Questa pratica assicura l'**high availability**, caratteristica fondamentale per la distribuzione di un servizio cloud. Questo motiva anche la presenza di due NAT gateway, che quindi consentirebbe la comunicazione alle risorse internet nel caso uno dei due fosse inutilizzabile.
C'è tuttavia una nota da fare per questa infrastruttura, poiché secondo questo ragionamento, l'istanza RDS potrebbe non essere raggiungibile se la zona B fallisse, ma tratterò questo punto nella sezione dedicata. Lo stesso vale per Amazon EC2 Auto Scaling.
È stato inoltre definito un **Internet gateway** collegato al vpc e che consente alle risorse l'accesso a internet. Sono state quindi definite le routing tables, in modo tale che le subnet private indirizzino il traffico verso i rispettivi **NAT gateway**, e che le subnet pubbliche indirizzino il traffico verso l'Internet gateway.

## 5.2 Amazon EC2 Auto Scaling (ASG)

Amazon EC2 Auto Scaling, anche conosciuto come Auto Scaling Group (ASG), è un servizio che, tramite i cosiddetti gruppi di autoscaling, ossia **raccolte di istanze EC2**, garantisce la disponibilità nel numero corretto di istanze e quindi permette di gestire automaticamente il carico sull'applicazione. I vantaggi del **ridimensionamento automatico** sono vari. Si veda la figura 5.2.
Garantisce una migliore tolleranza ai guasti poiché è in grado di rilevare quando un'istanza non è integra, terminarla e avviare un'istanza per sostituirla. Si può anche configurare Amazon EC2 Auto Scaling per utilizzare più zone di disponibilità, specificando le sottoreti in cui devono essere contenute le istanze. Se una zona di disponibilità diventa non disponibile, EC2 Auto Scaling può avviare istanze in un'altra per compensare.
EC2 Auto Scaling aiuta inoltre a garantire che l'applicazione disponga sempre della giusta quantità di capacità per gestire la domanda di traffico secondo analisi in real-time. Per questi motivi consente anche di ammortizzare i costi del servizio, che altrimenti sarebbero ben maggiori.
In fase di configurazione si possono specificare il numero minimo, massimo e desiderato di istanze in ciascun gruppo Auto Scaling oppure configurare un modello d'avvio specificando ad esempio l'ID AMI (*Amazon Machine Image*), il tipo di istanza, i gruppi di sicurezza ecc., uguale per tutte le istanze EC2 in esso contenute.
Inoltre è possibile configurare un gruppo in modo che si ridimensioni in base al verificarsi di condizioni specifiche o in base a una pianificazione, conosciute come *target scaling*. Il bilanciamento delle istanze avviene quindi in due modalità: secondo le zone di disponibilità e secondo la capacità richiesta del gruppo.

Per chiarire la potenzialità di questo servizio si consideri l'esempio che segue.
Si ipotizzi ad esempio ad una applicazione Web che consente di effettuare prenotazioni online per i posti disponibili di una sala studio, dal Lunedì alla Domenica. Durante il fine settimana si riscontra un utilizzo minimo di questa applicazione. Durante la metà della settimana invece, più studenti effettuano prenotazioni, quindi la domanda sull'applicazione aumenta in modo significativo.
Una prima soluzione sarebbe di distribuire il servizio su un numero sufficiente di istanze EC2 in modo che l'applicazione disponga sempre di capacità sufficiente per soddisfare la domanda. Lo svantaggio, tuttavia, è che ci sono giorni in cui l'applicazione non ha bisogno di così tanta capacità e che quindi aumenta il costo di mantenere l'applicazione in esecuzione.
Una seconda soluzione sarebbe quella di sfruttare il compromesso tra richieste e costi fornendo una capacità di calcolo che soddisfa la richiesta media complessiva. Tuttavia anche questa soluzione non è preferibile poiché l'esperienza finale dell'utente può rive-

larsi scadente quando le richieste sono elevate.
La soluzione migliore consiste quindi nell'utilizzare Amazon EC2 Auto Scaling perché scala le istanze secondo necessità evitando costi inutili.

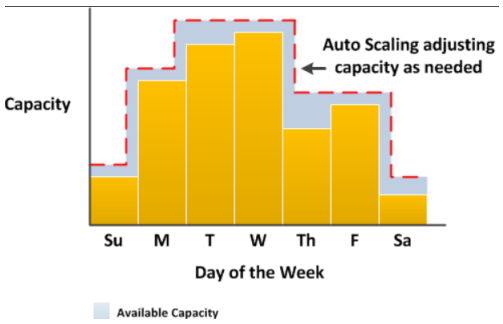


**Figura 5.2:** Ridimensionamento di Amazon EC2 Auto Scaling

## 5.3 Amazon Application Load Balancer (ALB)

L'Application Load Balancer fa parte degli Elastic Load Balancers. Questi servizi **distribuiscono automaticamente il traffico** in entrata su più destinazioni, come istanze EC2, in una o più zone di disponibilità, ridimensionando il sistema in base alle variazioni del traffico in entrata e adattandosi alla maggior parte dei carichi di lavoro. Questo permette quindi di aumentare la disponibilità dell'applicazione.
Un *listener*, collegato ad un sistema di bilanciamento, controlla le richieste di connessione dai client, utilizzando il protocollo e la porta configurati. Ad un listener sono collegate una o più *rules* (regole), che consistono in una priorità, e una o più azioni che si verificano quando le rispettive condizioni vengono verificate.
A quel punto il listener comunica con i *target groups*, i quali instradano le richieste a uno o più target registrati, come l'istanza EC2 Auto Scaling in questo caso, utilizzando il protocollo e il numero di porta specificati. Si veda la figura 5.3. Le istanze avviate vengono registrate automaticamente con il load balancer e le istanze terminate vengono automaticamente annullate dalla registrazione.

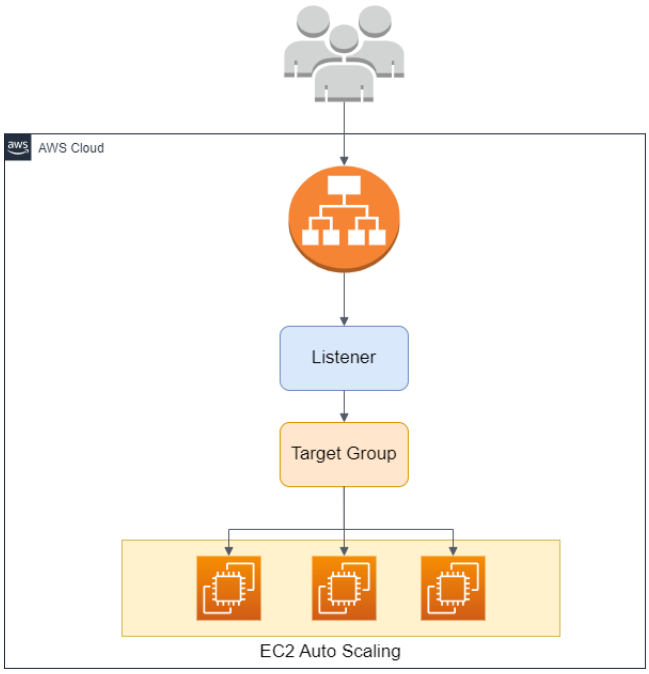


**Figura 5.3:** Funzionamento di Amazon Application Load Balancer

Elastic Load Balancing supporta quattro sistemi di bilanciamento del carico: Application Load Balancer, Network Load Balancer, Gateway Load Balancer e Classic Load Balancer, ognuno con caratteristiche diverse poiché operano su livelli diversi del modello **OSI** (*Open Systems Interconnection*).
Un Application Load Balancer funziona a livello di applicazione, il settimo livello del modello OSI. Il livello di applicazione viene utilizzato dal software dell'utente finale come il browser Web o il client di posta elettronica e fornisce protocolli che gli consentono di inviare e ricevere informazioni. Alcuni esempi di protocolli sono Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP) oppure Domain Name System (DNS).

## 5.4 Amazon Relational Database Service (RDS)

Amazon RDS è un servizio che consente di istanziare **database relazionali** nel cloud AWS, semplificandone la configurazione, l'uso e il dimensionamento. Nel presente progetto la configurazione è stata minima, tuttavia, siccome supporta differenti engine, nel modello templatizzato è possibile scegliere tra MySql, MariaDB e PostgreSQL; quest'ultimo è di default.
Una nota importante riguarda la disponibilità del database, menzionata nella sezione di vpc. Il database permette di implementare la funzionalità *multi-AZ*, soluzione che segue il concetto di high availability.
Esistono due tipologie di questa implementazione.
L'implementazione di *istanza* database Multi-AZ, fornisce un'istanza DB in standby, disponibile in una availability zone diversa da quella principale, impiegata in caso di

fallimento di quest'ultima. Tuttavia non consente di servire il traffico di lettura.
L'implementazione di *cluster* database Multi-AZ fornisce supporto per il fallimento accidentale di una availability zone ma può anche servire il traffico di lettura.

## 5.5 Amazon CodePipeline

Amazon CodePipeline è un servizio di **distribuzione continua** che permette di modellare, visualizzare e automatizzare i passaggi necessari per il rilascio continuo delle modifiche al software.
In ingegneria del software si parla di *Continuous Integration* e *Continuous Delivery*, conosciute anche con l'acronimo **CI/CD**, come le pratiche combinate di integrazione continua e distribuzione continua.
In particolare la **Continuous Integration** è una pratica di sviluppo software in cui i membri di un team utilizzano un sistema di controllo della versione e integrano quotidianamente il proprio lavoro nella stessa posizione, ad esempio un ramo principale, per rilevare gli errori di integrazione il più rapidamente possibile. Questa pratica è incentrata sulla creazione e sul test automatici del codice per limitare fin da subito l'insorgere di errori irreparabili.
La **Continuous Delivery** è invece una metodologia di sviluppo software in cui il processo di rilascio è automatizzato. Ogni modifica al software viene quindi creata e testata da una persona, un test automatizzato o una regola aziendale. A quel punto, se supera i test con successo, può essere immediatamente rilasciata in produzione, anche se non necessariamente tutte le modifiche devono essere rilasciate immediatamente.
CodePipeline permette quindi di applicare queste pratiche agili di sviluppo del software tramite i servizi Amazon.

In CodePipeline, una **pipeline** è un costrutto del flusso di lavoro che descrive come le modifiche al software passano attraverso un processo di rilascio. Ogni pipeline è costituita da una o più **stages** (fasi). Una fase è un'unità logica che contiene **actions** (azioni) che vengono eseguite sugli **artifacts** (artefatti) dell'applicazione.
Un'*azione* invece è un insieme di operazioni eseguite sul codice e configurate in modo che vengano eseguite nella pipeline in un punto specifico. Possono includere ad esempio un'azione di origine da una modifica del codice, un'azione per la distribuzione dell'applicazione alle istanze e così via.
Esempio di *artefatti* sono il codice sorgente del software, o più in generale l'input e l'output delle fasi, che possono essere utilizzati da altre fasi successive. Per salvare questi artefatti di output viene impiegato un bucket Amazon S3 legato e funzionale alla pipeline, come mostrato nel diagramma dell'infrastruttura.

Dopo questa introduzione esporrò di seguito il flusso della pipeline impiegata nel progetto. Faccio notare che lo sviluppo si è limitato a definire i servizi di pipeline e non ad integrarlo con un software vero e proprio, poiché usciva dallo scopo del progetto.

Come menzionato è stato creato un apposito bucket S3 per gli artefatti di output della pipeline.
**CodePipeline** è il servizio principale al cui interno si vanno a definire gli stages ( fasi) della pipeline. Ogni fase viene eseguita in un ordine prestabilito a partire da Amazon CodeCommit.
**CodeCommit** è un servizio di controllo del codice sorgente gestito, sicuro e ad elevata scalabilità che ospita repository Git privati. Esso ospita il codice sorgente del prodotto software e, a seguito di una modifica di questo, notifica CodePipeline che avvia l'esecuzione della pipeline. Il codice sorgente diventa quindi l'output di questa fase e diviene l'input della fase successiva, quella di Amazon CodeBuild.
**CodeBuild** è un servizio di compilazione completamente gestito nel cloud, che compila il codice sorgente, esegue i test d'unità e prepara artefatti pronti per essere distribuiti. L'attività di compilazione prepara l'ambiente di compilazione tramite le istruzioni contenute nel file *appspec.yaml* e compila l'applicazione in un contenitore virtuale con le istruzioni contenute nel file *buildspec.yaml*. Questi due file devono essere versionati nello stesso repository del codice sorgente.
Il codice compilato e testato viene successivamente archiviato nel bucket e ottenuto come input nella fase successiva, che distribuisce l'applicazione in un ambiente di produzione tramite il servizio Amazon CodeDeploy.
**CodeDeploy** è un servizio di distribuzione che automatizza la distribuzione dell'applicazione a istanze Amazon EC2, istanze locali, istanze di container(si veda l'ultimo progetto) e altri servizi, come EC2 Auto Scaling nel caso di questa infrastruttura.
È riportata nell'immagine 5.4 le fasi della pipeline nella console di AWS.

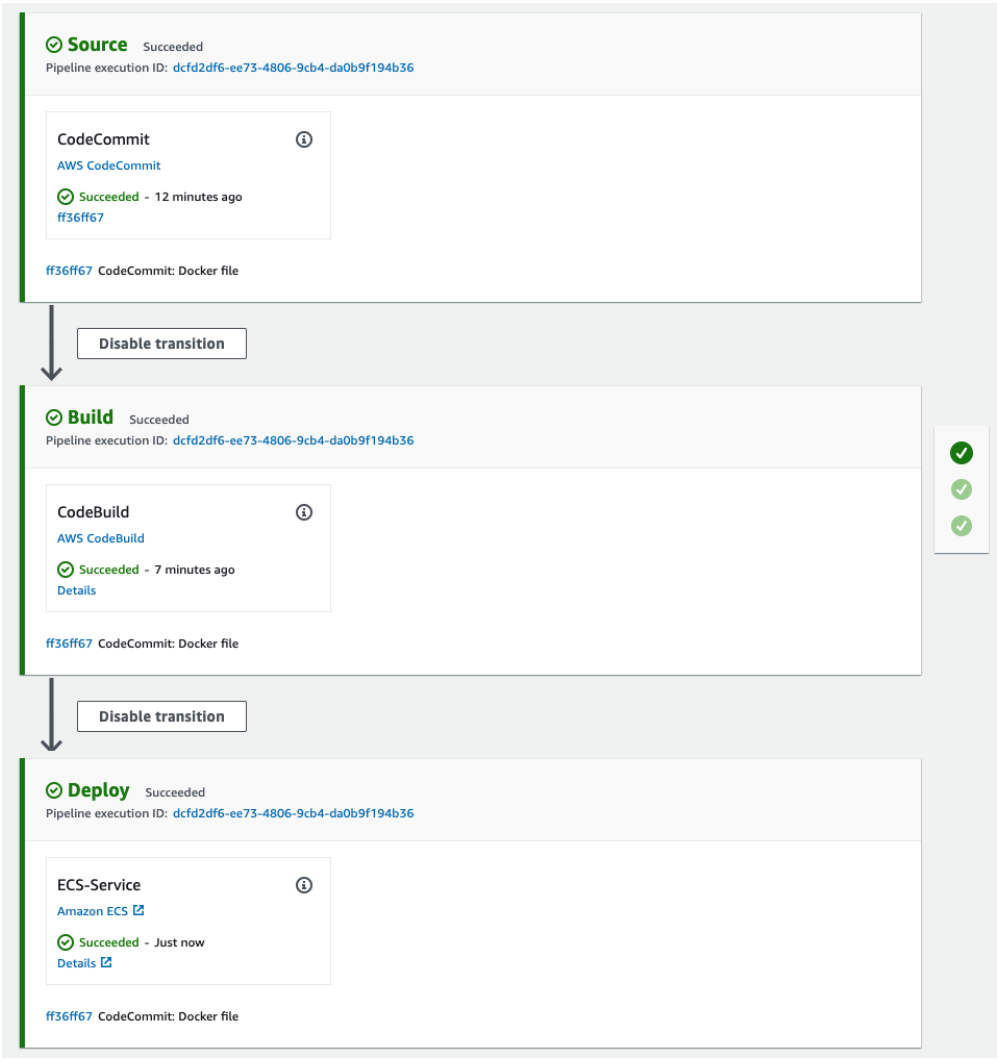


**Figura 5.4:** Esempio di CodePipeline nella console AWS

# Capitolo 6

# Template di infrastruttura con servizio Serverless

Nella seguente sezione esporrò il secondo progetto di infrastruttura in forma templatizzata. Il diagramma in figura 6.1 mostra le risorse AWS all'interno di AWS Cloud, ossia lo spazio all'interno dell'account AWS, e le relazioni, indicate dalle frecce, che sussistono fra loro. Il progetto è disponibile in GitHub nel repository pubblico raggiungibile a questo link

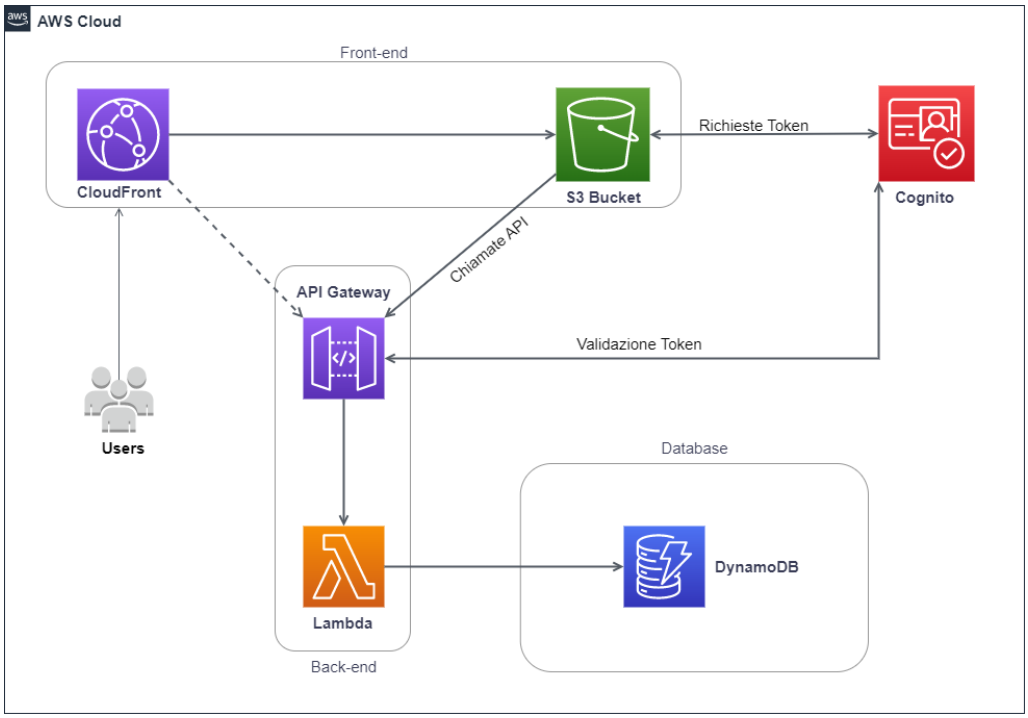


**Figura 6.1:** Diagramma infrastruttura con servizio Serverless

Quando si parla di ***serverless***, e in particolare di *serverless computing*, si intende un modello di sviluppo nel cloud che permette agli sviluppatori di creare ed eseguire applicazioni senza gestire i server. Anche se in questo modello i server vengono utilizzati

comunque, e sono completamente gestiti dal provider cloud, sono astratti dallo sviluppo delle app.
Dopo il rilascio, le infrastrutture serverless rispondono alle richieste e si adattano automaticamente in base alle diverse esigenze di scalabilità.
In questo caso vengono avviate solo quando un evento attiva l'esecuzione del codice, e il provider di cloud pubblico assegna dinamicamente le risorse. I vantaggi sono in termini di basso costo ed alta efficienza evitando inoltre agli sviluppatori le attività di routine e manutenzione manuale.
Il diagramma in figura mostra un'infrastruttura serverless che ruota attorno ad Amazon Lambda, il servizio che permette di applicare il modello appena descritto.

I principali servizi che compongono l'infrastruttura sono:

- **Amazon CloudFront**: servizio che accelera la distribuzione di contenuto Web statico e dinamico.
- **Amazon S3**: servizio di storage
- **Amazon Cognito**: servizio che fornisce autenticazione, autorizzazione e gestione degli utenti per le app Web e per dispositivi mobili
- **Amazon API Gateway**: servizio completamente gestito che semplifica per gli sviluppatori la creazione, la pubblicazione e la manutenzione delle API su qualsiasi scala.
- **Amazon Lambda**: servizio di calcolo che consente di eseguire il codice senza provisioning o gestione di server.
- **Amazon DynamoDB**: servizio di database NoSQL

Di seguito fornirò un'analisi dei componenti dell'infrastruttura e della sua logica, soffermandomi sui servizi non introdotti in precedenza.

## 6.1 Amazon CloudFront

Amazon CloudFront è un servizio che accelera la **distribuzione** di contenuto Web, sia statico che dinamico, distribuito tramite una rete mondiale di edge locations. Le *edge locations* sono data center AWS progettati per fornire servizi con la latenza più bassa possibile, sparsi in tutto il mondo. Sono più vicini agli utenti rispetto alle regions o alle availability zones, spesso nelle principali città, quindi le risposte possono essere molto rapide. Quando un utente richiede un oggetto tramite Amazon CloudFront, la richiesta viene instradata all'edge location che fornisce la latenza più bassa, per garantire le prestazioni migliori.

Se il contenuto è già nell'edge location, viene automaticamente distribuito. Altrimenti CloudFront lo recupera da un'origine predefinita come un bucket S3 o un server HTTP. L'uso della rete AWS, e in particolare della rete dorsale delle edge locations, riduce drasticamente il numero di reti attraverso le quali le richieste devono transitare e di conseguenza migliora le prestazioni.

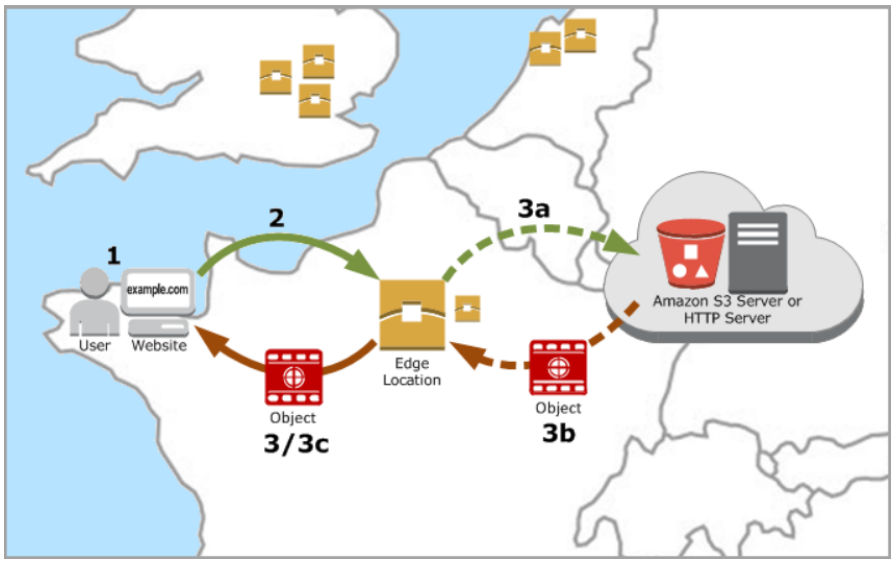


**Figura 6.2:** Funzionamento di Amazon CloudFront

L'immagine 6.2 descrive semplicemente ciò che avviene quando un utente richiede un contenuto. (1) Egli accede ad un sito Web e richiede un oggetto. (2) La richiesta viene quindi inoltrata da CloudFront alla edge location a latenza più bassa. A questo punto CloudFront controlla se è già presente nella cache: (3) se è già presente lo ritorna direttamente all'utente; (3a) altrimenti inoltra la richiesta ad esempio al bucket S3, (3b) questo risponde ritornando l'oggetto richiesto alla adge location che lo salva a sua volta nella cache per future richieste, (3c) e quindi ritorna l'elemento all'utente.
Nel progetto dell'infrastruttura è stata creata un'istanza di CloudFront tramite la classe *Distribution*, collegata ad un bucket S3 come origine di un'ipotetica applicazione Web.

## 6.2 Amazon Cognito

Amazon Cognito è un servizio che fornisce **autenticazione**, **autorizzazione** e **gestione degli utenti** per le app Web e per dispositivi mobili. Gli utenti possono accedere direttamente con username e password, oppure tramite servizi di terze parti, ad esempio Facebook, Amazon, Google o Apple.
I due componenti principali di Cognito sono le user pools e le identity pools.
Una **user pool** è una directory di utenti in Cognito, tramite la quale gli utenti dell'applicazione vi possono accedere direttamente oppure indirettamente, attraverso un provider di identità (IdP) di terza parte. Dopo una corretta autenticazione, l'appli-

cazione Web o per dispositivi mobili riceverà dei token della user pool da Amazon Cognito.Si veda la figura 6.3.
Una **identity pool** permette invece agli utenti di ottenere credenziali AWS temporanee per accedere ai servizi, come Amazon S3 o API Gateway.

I pool di identità supportano utenti guest anonimi, nonché tramite IdP di terze parti.

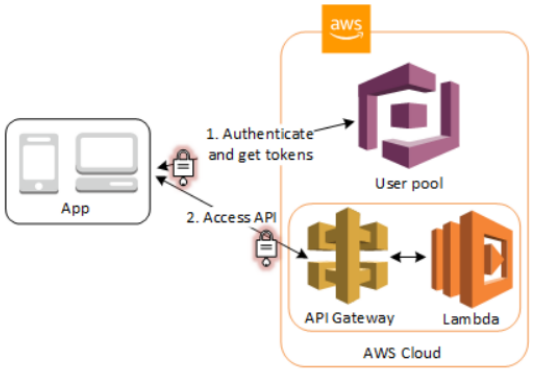


**Figura 6.3:** Funzionamento di Amazon Cognito

I **token**, forniti all'applicazione a seguito di un'autenticazione andata a buon fine, consentono quindi di essere successivamente scambiati con altri servizi. Questi servizi validano tali token creando credenziali temporanee, se ne è consentito l'accesso. La figura mostra quanto descritto ed illustra in maniera differente ciò che avviene nell'infrastruttura.
Il flusso di una richiesta client nell'infrastruttura è il seguente: la richiesta di un oggetto da parte di un utente passa ad Amazon CloudFront. Questo verifica che sia nella cache, ma supponiamo che lo richieda al bucket S3, il quale deve prima autenticarsi presso Amazon Cognito. Questo gli ritorna i token a seguito dell'autenticazione e quindi S3 può a questo punto effettuare chiamate all'API Gateway (che verrà descritto successivamente). Questo valuterà il permesso per eseguire la richiesta verificando i token consegnati da S3 tramite Cognito e, in caso di esito positivo, inoltrerà la richiesta ad Amazon Lambda che si occuperà del resto.

## 6.3 Amazon API Gateway

Prima di introdurre questo servizio riporto una descrizione di cosa sono le API e per quale scopo sono impiegate.
Il termine **API**, acronimo di *Application Programming Interface*, indica quei meccanismi che consentono a due componenti software di comunicare tra loro usando una serie di definizioni e protocolli. Si può pensare all'interfaccia come ad un **contratto tra due applicazioni**, che definisce come queste comunicano tra loro usando richieste

e risposte. La rispettiva documentazione dell'API contiene informazioni su come gli sviluppatori devono strutturare tali richieste e risposte.
Per questa caratteristica, le API permettono ai propri prodotti o servizi di comunicare con altri prodotti o servizi, anche senza sapere come sono stati implementati, semplificando così lo sviluppo delle app e risparmiando tempo e denaro. Inoltre, semplificano l'integrazione di nuovi componenti applicativi nell'architettura esistente e promuovono la collaborazione tra team. Grazie alle API, e in particolare a quelle pubbliche come ad esempio le API di Google Maps, è quindi possibile collegare con facilità l'infrastruttura mediante lo sviluppo di applicazioni cloud e condividere i dati con i clienti e con altri utenti esterni.
La tipologia di API più utilizzata e diffusa nel Web è l'**API REST**. REST sta per "Representational State Transfer", e definisce una serie di funzioni GET, POST, PUT, PATCH e DELETE che i client possono usare per accedere ai dati del server, scambiandosi i dati tramite il protocollo HTTP. La loro caratteristica principale è che sono stateless, ossia il server non salva i dati del client tra le richieste.
Esiste poi un'altra tipologia, L'**API WebSocket**, che supporta la comunicazione a due vie (full-duplex) tra client e server e usa oggetti JSON per trasferire i dati. Inoltre, il server può inviare messaggi di callback ai client connessi, migliorando l'efficienza rispetto all'API REST.

Amazon API Gateway permette quindi agli sviluppatori la possibilità di creare, pubblicare, gestire, monitorare e proteggere API REST, HTTP e WebSocket. Queste possono essere usate nelle proprie applicazioni oppure per renderle disponibili agli sviluppatori di terze parti. Riporto ora la descrizione di un'API REST in API Gateway; questa include risorse e metodi.
Una **risorsa** è un'entità logica a cui un'app può accedere tramite un percorso della risorsa. Un **metodo** corrisponde a una richiesta API inviata dall'utente e alla risposta restituita all'utente.
Ad esempio, */incomes* potrebbe essere il percorso di una risorsa che rappresenta il guadagno dell'utente e questa risorsa può avere diversi metodi per operare su di essa. Ad esempio POST */incomes*, potrebbe aggiungere il reddito guadagnato mentre GET /incomes potrebbe eseguire una query relativa alle spese sostenute.
In API Gateway, il front-end è incapsulato dalle *richieste di metodo* e dalle *risposte di metodo*. L'API si interfaccia con il back-end tramite le *richieste di integrazione* e le *risposte di integrazione*, come si può vedere nell'immagine 6.4 dalla console AWS.

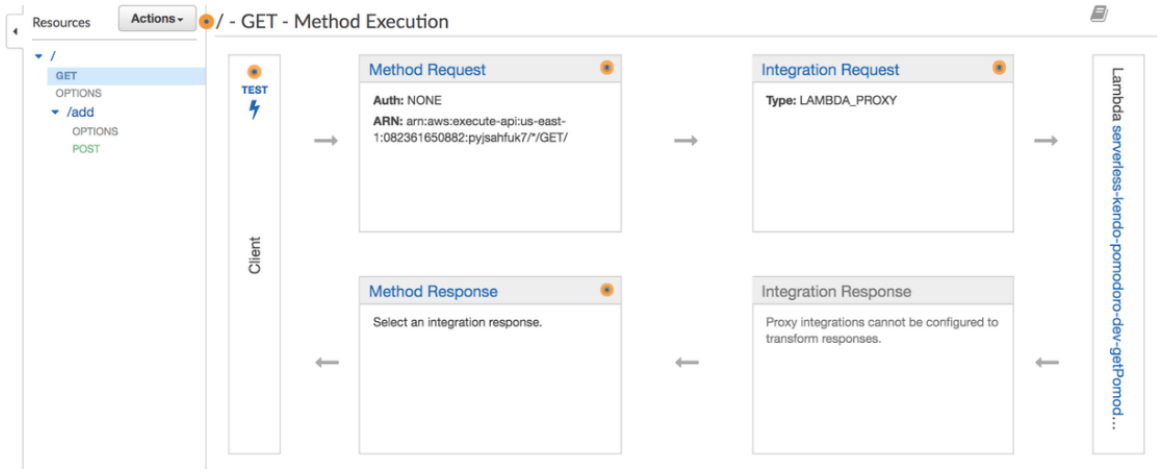


**Figura 6.4:** Esempio di API Gateway nella console AWS

Nel progetto di stage, e in particolare nello stack di API Gateway, la forma templatizzata mi ha consentito di poter definire i vari metodi e risorse nel file cdk.json e quindi aggiungerli al gateway tramite un iterazione su questi. Faccio tuttavia notare che, anche in questo caso come nella precedente infrastruttura, non ho sviluppato alcuna applicazione e quindi non ho definito alcuna API perchè non era l'obiettivo dello stage e usciva dallo scopo di progettazione e sviluppo di una infrastruttura, in quanto, in ambiente aziendale come in zero12, sono altri sviluppatori ad occuparsi di esse.

## 6.4 Amazon Lambda

Veniamo ora al cuore del servizio che consente di applicare il concetto di **serverless** introdotto all'inizio. Faccio notare che l'infrastruttura non ha una propria vpc, in quanto non strettamente necessaria per Lambda, poiché utilizza una propria vpc all'esterno dell'infrastruttura.
Amazon Lambda è un **servizio di calcolo** che consente di eseguire il codice senza provisioning o gestione di server. Tutte le risorse di elaborazione, compresa la manutenzione del server e del sistema operativo, il provisioning e la scalabilità automatica della capacità sono gestite da Amazon, lasciando quindi agli sviluppatori la possibilità di occuparsi del resto.
Il codice viene organizzato in **funzioni Lambda**, configurabili utilizzando la console Lambda, l'API Lambda o AWS CloudFormation, che vengono eseguite solo quando è necessario. Il servizio permette di eseguire più funzioni in parallelo e si dimensiona automaticamente, da poche richieste al giorno a migliaia al secondo, addebitando solo i costi di quando esso è in esecuzione. È possibile richiamare le funzioni utilizzando l'API di Lambda oppure in risposta agli eventi di altri servizi AWS.
Fornisce runtime, come Node JS, Python o Java, che supportano un set standard di caratteristiche, in modo da poter passare facilmente da linguaggi a framework a secon-

da delle esigenze.
In questo scenario infrastrutturale, Lambda riceve le richieste dall'API Gateway, che quindi andrà ad eseguire tali funzioni operando sulle risorse di DynamoDB, servizio che descriverò successivamente, e Lambda a quel punto risponderà alla richiesta ritornando l'oggetto chiamato.
Come si può vedere nel seguente snippet di codice del progetto, lo stack di Lambda, nominato *lambda.ts*, definisce un ciclo che istanzia la classe *Function* che definisce una particolare funzione configurata tramite la forma templatizzata. Faccio notare in particolare le proprietà richieste, tra cui il *runtime*, l'*handler*, ossia il nome del metodo all'interno del codice che Lambda chiamerà quando verrà eseguito e *code*, che contiene un riferimento ad un percorso per il codice da eseguire. In questo progetto, è stata configurata una sola funzione JavaScript, definita nella cartella *lambdaFn*.

```
lambdaConfig.forEach((lambda) => {
    const newLambda= new Function(this, lambda.id, {
      functionName: `${prefix}-${lambda.id}`,
      runtime: new Runtime(lambda.runtime),
      handler: lambda.handler,
      code: Code.fromAsset(path.join(__dirname, lambda.codePath)),
    })
[...]
}
```

**Listing 6.1:** Istanziazione di funzioni Lambda dal progetto

Il file *cdk.json* contiene invece tali parametri, che vanno presi come esempio, e che sono contenuti come oggetti in un array, nel quale è possibile aggiungere tutte le funzioni volute. Anche qui si deve assumere che la funzione non esegue nulla di rilevante ma è stata inserita per completezza.

```
"lambda": [
          {
            "id": "FirstLambda",
            "runtime": "nodejs 16.x",
            "handler": "functionLambda.firstHandler",
            "codePath": "/lambdaFn",
            "grantedToDynamo": true,
            "grantWriteToDynamo": true
          },
]
```

**Listing 6.2:** Esempio di configurazione di funzione Lambda in cdk.json

## 6.5 Amazon DynamoDB

Amazon DynamoDB è un servizio di **database NoSQL**. Questo termine è usato per descrivere i sistemi di database non relazionali altamente disponibili, scalabili e ottimizzati per prestazioni elevate. Anziché il modello relazionale come per Amazon RDS, DynamoDB utilizza modelli alternativi per la gestione dei dati, ad esempio, coppie chiave-valore o archiviazione di documenti. Inoltre consente di scaricare gli oneri di installazione, gestione, configurazione, replica o dimensionamento del cluster. Tutti i dati vengono replicati automaticamente in più zone di disponibilità in una regione AWS in modo da fornire elevata disponibilità e durabilità dei dati.
In DynamoDB, le tabelle, le voci e gli attributi sono i componenti principali da utilizzare. Una *table* è una raccolta di *items* e ogni *item* è una raccolta di *attributes*. DynamoDB utilizza le chiavi primarie per identificare in modo univoco ciascun elemento in una tabella. Amazon DynamoDB supporta un linguaggio di query open source compatibile con SQL che semplifica la query dei dati in modo efficiente, indipendentemente da dove o in quale formato sono memorizzati.
Una applicazione, per accedere a DynamoDB, necessita prima dell'autenticazione e in seguito delle autorizzazioni per effettuare richieste.
A seguito di ciò, l'applicazione invia le richieste HTTP a DynamoDB, che contengono il nome dell'operazione da eseguire insieme ai parametri. DynamoDB restituisce quindi una risposta HTTP contenente i risultati dell'operazione, oppure un codice d'errore HTTP in caso di problemi.

# Capitolo 7

# Template di infrastruttura con servizio Container

Nella seguente sezione esporrò il terzo progetto di infrastruttura in forma templatizzata. Il diagramma in figura 7.1 mostra le risorse AWS all'interno di AWS Cloud, ossia lo spazio all'interno dell'account AWS, e le relazioni, indicate dalle frecce, che sussistono fra loro. Il progetto è disponibile in GitHub nel repository pubblico raggiungibile a questo link.

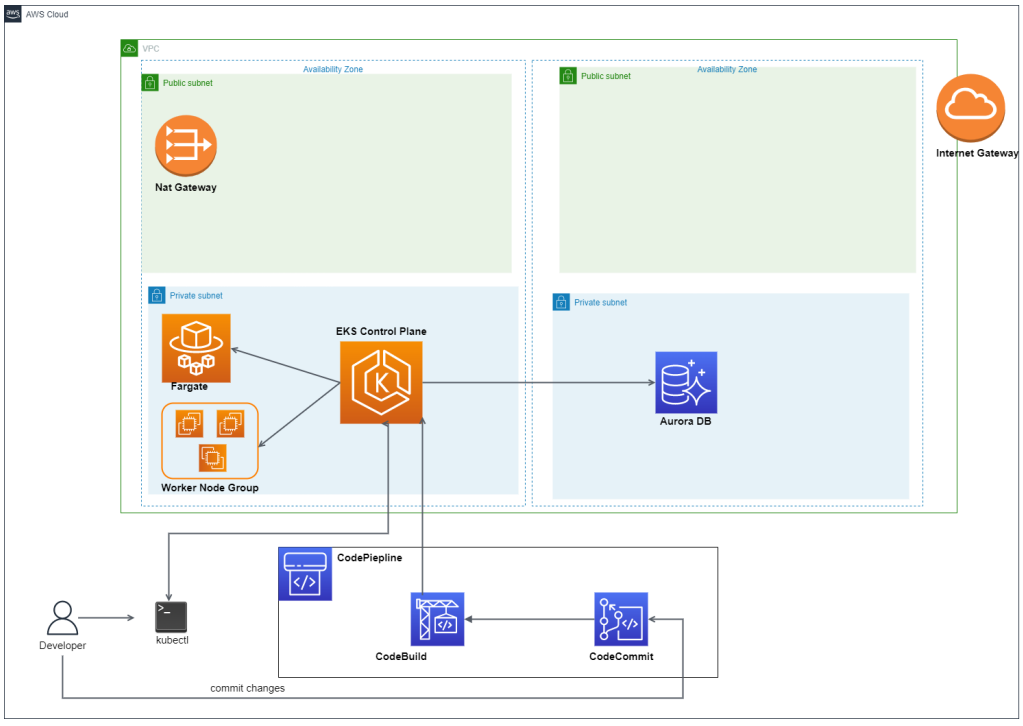


**Figura 7.1:** Diagramma infrastruttura con servizio Container

Prima di descrivere l'infrastruttura sviluppata dell'ultimo progetto, fornirò una descrizione di cosa sono i container e per quale scopo vengono utilizzati.

I **container** sono unità eseguibili di software in cui viene impacchettato il codice del prodotto software, assieme alle sue librerie e dipendenze, con modalità comuni in modo da poter essere eseguito in ogni ambiente, da un data center privato al cloud pubblico o anche sul laptop di uno sviluppatore. La containerizzazione consente ai team di sviluppo di spostarsi velocemente, eseguire il deployment del software in modo efficiente e operare su una scala senza precedenti.
I container consentono di virtualizzare le risorse di CPU, memoria, archiviazione e rete a livello di sistema operativo, offrendo agli sviluppatori una visualizzazione del sistema operativo logicamente isolata da altre applicazioni.
Inoltre, i container sono piccoli, veloci e portatili perché possono sfruttare le funzioni e le risorse del sistema operativo. Al contrario, le *Virtual Machine* (VM) effettuano la virtualizzazione a livello hardware e hanno pertanto bisogno di includere un sistema operativo guest in ogni istanza, il che li rende meno efficaci da distribuire. Questi vantaggi hanno reso i container una delle forme più utilizzate per la distribuzione di prodotti software.
Il diagramma in figura mostra un'infrastruttura con servizi di containerizzazione che ruota attorno ad **Amazon EKS**, servizio che permette di eseguire e gestire automaticamente i container, che descriverò in seguito.

I principali servizi che compongono l'infrastruttura sono:

- **Amazon VPC**: servizio che definisce la componente di network e altri servizi funzionali ad essa come NAT Gateway e Internet Gateway. Per questa infrastruttura non mi soffermerò su questa parte in quanto è stata ripresa dal primo progetto di infrastruttura con servizio di pipeline.

- **Amazon EKS**: servizio Kubernetes che semplifica l'esecuzione di Kubernetes su AWS, che quindi agisce come gestore per i container.

- **Amazon Fargate**: motore di calcolo serverless per l'esecuzione di container e che funziona con Amazon EKS.

- **Amazon Aurora**: servizio che fornisce un motore del database relazionale completamente gestito basato su Amazon RDS.

- **Amazon CodePipeline**: servizio per creare una pipeline di CI/CD. Non mi soffermerò su questo servizio in questa sezione in quanto offre le stesse funzionalità descritte per il primo progetto. L'unica nota che intendo riportare riguarda la mancanza di Amazon CodeDeploy, omesso per questioni di semplicità e per il fatto che anche Amazon CodeBuild, come si può vedere in figura, può effettuare il rilascio nelle istanze Kubernetes.

## 7.1 Docker e Kubernetes

### Docker

Come descritto, i container sono la forma di distribuzione da preferire per un'applicazione containerizzata e per fare ciò vengono offerte diverse piattaforme per la loro creazione.
Docker è la piattaforma software più popolare e utilizzata che permette di creare, testare e distribuire applicazioni con la massima rapidità, raccogliendo il software in unità standardizzate, ossia i container, e che include tutto il necessario per la loro corretta esecuzione, incluse librerie, strumenti di sistema, codice e runtime.

I container Docker sono particolarmente rilevanti per l'esecuzione di architetture di microservizi e incentivano l'impiego delle pratiche di Continuous Integration e Continuous Delivery (CI/CD) tramite la distribuzione del codice con l'utilizzo di pipeline, come Amazon CodePipeline.
In particolare Docker collabora con AWS per aiutare gli sviluppatori a velocizzare la transizione di applicazioni moderne al cloud, semplificando la distribuzione dei manufatti Docker tramite Amazon EKS e Amazon Fargate, servizi impiegati nell'infrastruttura del progetto che descriverò successivamente. Infatti, poiché si tratta di un sistema operativo per container, questo virtualizza il sistema operativo di ogni server, fornendo semplici comandi con cui creare, avviare o interrompere i container.

### Kubernetes

Molte aziende, nonostante la semplicità di gestione del singolo container, si sono dovute scontrare con la complessità della gestione di centinaia o anche migliaia di container su un sistema distribuito. A questo proposito è cominciato a nascere un insieme di piattaforme per l'orchestrazione di container, come un modo per gestire grandi volumi di container lungo tutto il loro ciclo di vita. Tra le più importanti ed utilizzate troviamo Kubernetes, un progetto open source introdotto da Google che si propone come ***container orchestrator***.

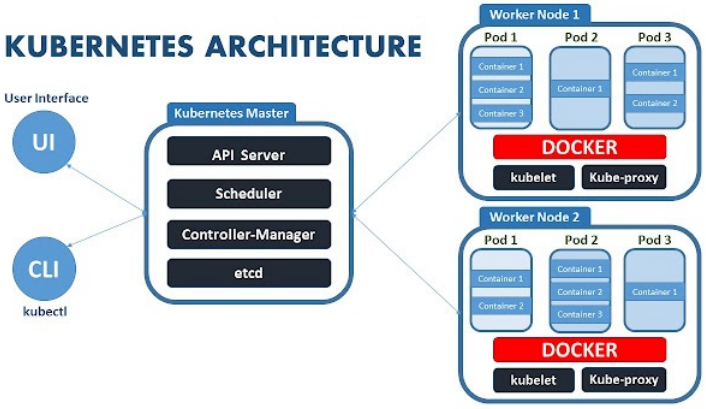


**Figura 7.2:** Architettura di Kubernetes

Kubernetes automatizza le operazioni di gestione dei container, assicurando la continuità dei servizi e l'allocazione efficiente delle risorse. In particolare può distribuire il carico di lavoro tra più container durante i picchi di traffico, per allocare automaticamente le risorse computazionali ad un livello adeguato, effettuare rollback automatizzati e gestire informazioni critiche di sicurezza come credenziali o token delle applicazioni containerizzate.
In Kubernetes ci sono alcuni componenti fondamentali alla base del suo funzionamento che richiamerò in seguito. L'immagine 7.2 mostra sinteticamente il funzionamento e i componenti principali di Kubernetes.

Il **Master** è la macchina che controlla i nodi di Kubernetes e permette di allocare dinamicamente i container in base alle risorse disponibili e ai parametri predefiniti.
Il master a sua volta gestisce i **Worker Nodes** che vengono raggruppati in **cluster** e che rappresentano le macchine virtuali o fisiche che hanno il compito di eseguire le attività assegnate. In essi troviamo Docker, che agisce da container runtime, ossia un componente software che permette di eseguire container Docker.
**Kubelet** invece si tratta di un agente software eseguito sui nodi che, ricevendo gli ordini dalla macchina master, avvia la gestione dei container secondo le istruzioni.
All'interno dei worker nodes troviamo i **Pods**, i quali indicano l'insieme dei container che risiedono sullo stesso nodo, condividendo, oltre all'indirizzo IP, le risorse di calcolo, storage e rete
Infine menziono **kubectl**, che si tratta di uno strumento da riga di comando (CLI) che consente di eseguire comandi sui cluster Kubernetes, ad esempio per distribuire applicazioni, ispezionare e gestire le risorse del cluster e visualizzare i log.

## 7.2 Amazon EKS

### 7.2.1 Introduzione

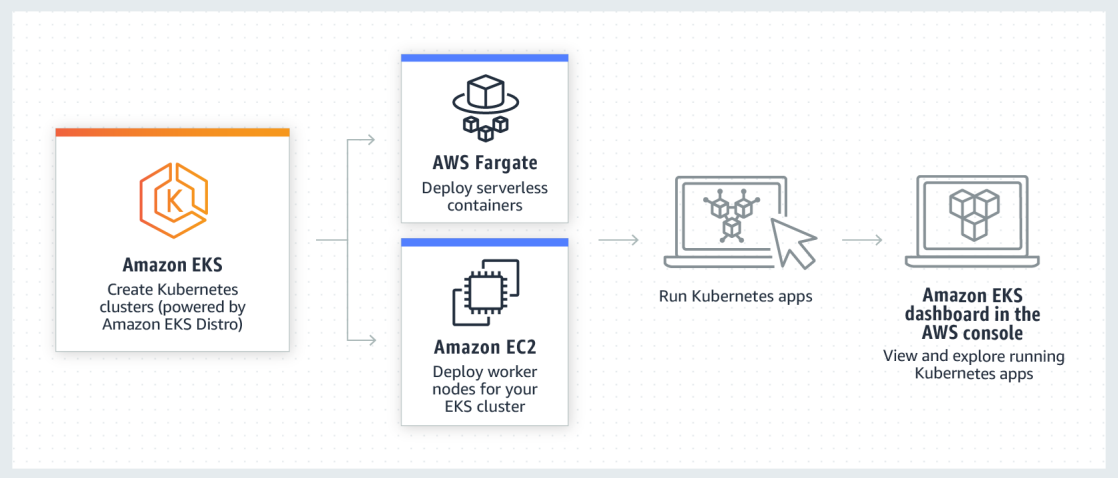


**Figura 7.3:** Amazon EKS

Amazon Elastic Kubernetes Service (Amazon EKS) è un servizio che permette di utilizzare Kubernetes in AWS eliminando la necessità di installare, mantenere ed utilizzare il control plane del nodo master ed i nodi ad esso relativi (vedi figura 7.3). Assicura elevata disponibilità effettuando il dimensionamento in più zone di disponibilità, scala automaticamente le istanze del piano di controllo in base al carico sostituendo quelle non integre ed esegue le versioni sempre aggiornate del software Kubernetes in modo da poter usare tutti gli strumenti della community Kubernetes.
In Amazon EKS, quando si parla di **cluster**, ci si riferisce a due componenti: l'**EKS Control Plane**, ossia il piano di controllo ed i **worker nodes** ad esso registrati, ossia i nodi di lavoro sui quali vengono eseguiti i Pods.
Il piano di controllo, sul quale viene eseguito il software Kubernetes, viene gestito in un account da AWS e l'API Kubernetes è esposta tramite l'endpoint associato al cluster, che gli consente pertanto di sfruttare le funzionalità Kubernetes. Per via della sua elevata disponibilità, dispone di un Elastic Load Balancer gestito da AWS.
I nodi di lavoro invece vengono eseguiti nell'account AWS e si connettono al piano di controllo tramite l'endpoint del cluster. I nodi possono essere creati in tre differenti forme: *nodi gestiti*, *nodi autogestiti* e *Amazon Fargate*. In particolare, come si può vedere anche dal diagramma, sono state impiegate le ultime due tipologie, poiché il piano di controllo può gestire più nodi di tipologie differenti e pertanto pianificare i Pods su qualsiasi combinazione di questi.

### 7.2.2 Worker nodes e Amazon Fargate

I gruppi di nodi gestiti, o **managed nodes**, eseguono automaticamente il provisioning e la registrazione delle istanze EC2 per conto dell'utente, a posteriori di una iniziale configurazione personalizzata. Ogni nodo viene assegnato come parte di un gruppo di Amazon EC2 Auto Scaling. In questo modo EKS crea e aggiorna automaticamente i nodi del piano dati, consentendo agli utenti di creare, aggiornare o terminare le istanze come poche e semplici operazioni.

I gruppi di nodi autogestiti, o **self-managed nodes**, richiede all'utente la creazione e la gestione dei nodi tramite configurazione manuale o tramite i modelli AWS CloudFormation. Un gruppo di nodi, come per il precedente, è costituito da più istanze EC2 di un gruppo di Auto Scaling. Quando le istanze EC2 sono in esecuzione, si uniscono al piano di controllo Kubernetes. Quindi, il piano di controllo Kubernetes assegna i Pods ai nodi ed esegue i container. Inoltre, l'utente ha l'onere di gestire l'aggiornamento del sistema operativo e della versione Kubernetes dei nodi del piano.
Nel progetto ho configurato questa tipologia di nodi specificando la tipologia delle istanze, il loro numero desiderato e la capacità dello storage, sempre in forma teamplatizzata. Nel diagramma dell'infrastruttura corrisponde all'elemento *Worker Node Group*.

Infine troviamo **Amazon Fargate**, anch'esso implementato per essere gestito dal piano di controllo di EKS. Fargate è una servizio che fornisce capacità di calcolo on-demand e di dimensioni adeguate per i container, che funge quindi da motore di calcolo serverless. Pertanto non è più necessario effettuare la creazione e la configurazione dei nodi e delle istanze ad essi relative. Viene quindi eliminata la necessità per l'utente di scegliere i tipi di server o di decidere come e quando dimensionare i gruppi di nodi in quanto è AWS ad occuparsene.

### 7.2.3 Componenti Kubernetes

Nel progetto e in particolare nello stack di EKS, oltre ad aver configurato il master control plane, i nodi autogestiti e Amazon Fargate, ho implementato i componenti che permettono di definire i Pods e le relative configurazioni. In particolare, tramite la forma a template ho configurato i componenti fondamentali, ma ne esistono molti altri per una configurazione più dettagliata: *Deployments, Service, ConfigMap e Secret*, che descrivo di seguito. Solitamente questi vanno definiti come file YAML e che vanno applicati successivamente alla creazione di un cluster per definirne la configurazione. Tramite CDK è possibile definirli all'interno del codice che verranno applicati automaticamente in fase di rilascio.

Un **Deployments** indica a Kubernetes come creare o modificare una o più istanze

di Pods che contengono un'applicazione containerizzata. Permette quindi di scalare orizzontalmente ed in modo efficiente il numero di Pods di replica, per il quale è possibile specificare il numero desiderato, abilitare l'implementazione del codice aggiornato in modo controllato o eseguire il rollback a una versione di distribuzione precedente, se necessario. Il controller di deployments Kubernetes monitora continuamente lo stato dei Pods e dei nodi e può apportare modifiche in tempo reale, come l'avvio delle istanze pod o la sostituzione di un pod guasto. Ciò consente di risparmiare tempo e mitigare gli errori automatizzando il lavoro e le funzioni manuali e ripetitive. Inoltre ogni Deployment e ogni gruppo di Pod, creati dallo stesso Deployment, possiedono una label che consente loro di essere identificati dalle altre risorse in Kubernetes. Questo vale anche per ogni altro componente

Un **Service** è invece un altro componente che fornisce l'accesso di rete a un set di Pods in Kubernetes. Un Service seleziona i Pods in base alle loro labels. Quando viene effettuata una richiesta di rete al Service, questo seleziona tutti i Pods nel cluster che corrispondono al selettore e ne sceglie uno a cui inoltrare la richiesta. La differenza tra Deployment e Service è che il primo è responsabile del mantenimento in esecuzione di un set di pod mentre il secondo è responsabile dell'accesso di rete a un set di pod.

Un **ConfigMap** è un componente usato per memorizzare dati non riservati in coppie chiave-valore. I Pods possono utilizzare le ConfigMaps come variabili d'ambiente, argomenti da riga di comando, o come file di configurazione all'interno. La ConfigMap permette di disaccoppiare le configurazioni specifiche per ambiente dalle immagini del container, cosicché le tue applicazioni siano facilmente portabili.

Un **Secret** invece è un componente che viene impiegato per memorizzati dati sensibili come password, token e chiavi SSH nei cluster, riducendo il rischio di esposizione dei dati a utenti non autorizzati, preferibile quindi in tal caso a ConfigMap, con il quale tuttavia condivide varie somiglianze.

## 7.3 Amazon Aurora

Amazon Aurora è un servizio che fa parte del database gestito Amazon Relational Database Service (Amazon RDS). Impiega un motore del **database relazionale** compatibile con MySQL e PostgreSQL, compatibile con il codice, gli strumenti e le applicazioni che vengono utilizzate con i database MySQL e PostgreSQL. In particolare Aurora ne migliora da tre fino a cinque volte le prestazioni poiché include un sottosistema di storage ad alte prestazioni. Le operazioni di gestione implicano cluster interi di server di database sincronizzati mediante operazioni di replica anziché singole istanze database. Le funzioni automatiche semplificano e rendono più conveniente dal punto di vista di

costi, configurazione, utilizzo e dimensionamento.

Un *cluster* database di Amazon Aurora è composto da una o più istanze e da un *volume del cluster* che gestisce i dati per tali istanze. Un volume del cluster Aurora è uno storage di database virtuale che si estende su più zone di disponibilità, ciascuna delle quali include una copia dei dati del cluster database. Ciascun cluster è composto quindi da due tipi di istanze. L'istanza database *primaria* supporta operazioni di lettura e scrittura ed esegue tutte le modifiche ai dati e si presenta come l'unica istanza di questo tipo in un cluster.
Le istanze di *replica* Aurora, che supporta fino ad un massimo di 15 istanze per cluster, si connettono allo stesso volume di storage dell'istanza primaria e supporta solo operazioni di lettura. Inoltre possono essere distribuite in diverse zone di disponibilità per contrastare possibili fenomeni di failover e quindi assicurare elevata disponibilità.
L'immagine 7.4 illustra quanto descritto.

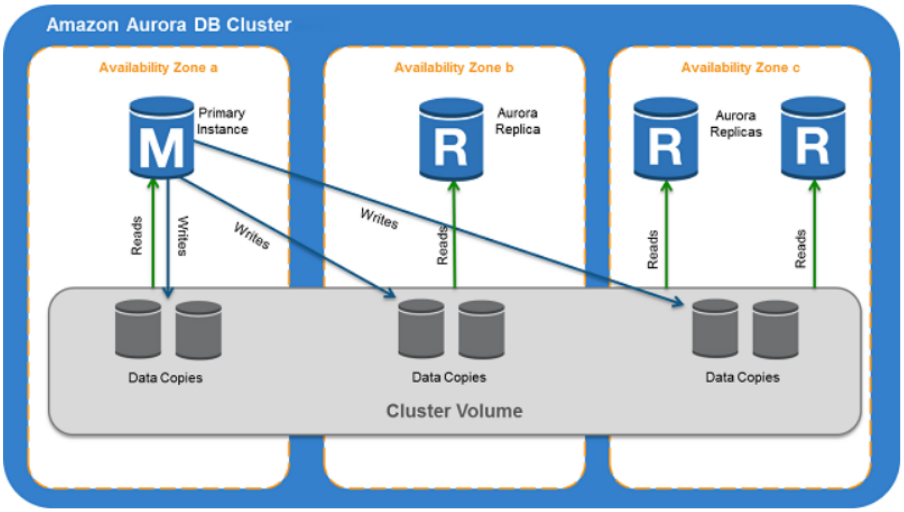


**Figura 7.4:** Amazon Aurora

# Capitolo 8

# Distribuzione

Nella seguente sezione descriverò i comandi che ho utilizzato da linea di comando nella directory di ogni progetto per effettuare il rilascio delle infrastrutture in AWS alla fine della fase di codifica di ognuno.
Tutti gli stack che vengono rilasciati, e quindi le risorse ad essi relative, possono essere visualizzabili nella console di AWS CloudFormation.
Faccio notare che nel progetto ho configurato, nel file *package.json*, dei comandi brevi che spiegherò di seguito e che richiamano i comandi nella loro completezza con tutte le proprietà necessarie. Questi comandi (figura 8.1) per essere invocati va utilizzato *npm run*, ad esempio: *npm run cdk-deploy-stack*

```
14    "cdk-deploy-dev-all": "cdk context --clear && cdk deploy \"*\" -c config=dev  --require-approval=never",
15    "cdk-deploy-stack": "cdk context --clear && cdk deploy $npm_config_stack -c config=dev --require-approval=never",
16
17    "cdk-diff-dev":"cdk context --clear && cdk diff -c config=dev",
18
19    "cdk-destroy-stack-dev": "cdk context --clear && cdk destroy $npm_config_stack -c config=dev ",
20    "cdk-destroy-dev-all": "cdk context --clear && cdk destroy \"*\" -c config=dev --require-approval=never"
21
```

**Figura 8.1:** Comandi cdk per la distribuzione

## 8.1 Bootstrapping

Il bootstrap è il processo di **provisioning delle risorse** per AWS CDK prima di poter effettuare il rilascio in un ambiente AWS. Queste risorse includono un bucket Amazon S3 per l'archiviazione di file e ruoli IAM che concedono le autorizzazioni necessarie per eseguire le distribuzioni. Le risorse richieste sono definite in uno stack AWS CloudFormation che di solito è denominato ***CDKToolkit***. Come qualsiasi stack, viene visualizzato nella console AWS CloudFormation una volta distribuito.
Il bootstrap deve essere eseguito sempre prima di effettuare un rilascio affinché AWS sappia dove rilasciare la distribuzione.

Il comando utilizzato è: ***cdk bootstrap*** *aws://ACCOUNT-NUMBER/REGION*. In particolare:

- *ACCOUNT-NUMBER*: rappresenta l'id dell'account sul quale si vuole distribuire.
- *REGION*: rappresenta il codice della region nella quale si vuole distribuire.

## 8.2 Confronto di versioni

Durante lo sviluppo di una infrastruttura è possibile andare incontro a diverse aggiunte, aggiornamenti o eliminazione di proprietà, risorse o interi stack. Ad ogni modo, prima di effettuare il rilascio è buona pratica effettuare un confronto tra la versione precedente e quella attuale, ossia visualizzare le modifiche che AWS CloudFormation apporterà al modello e che andrà a generare l'infrastruttura.
Il comando è ***cdk diff***, che va a listare tutte le risorse e le relative modifiche.
In particolare nel progetto ho utilizzato *npm run cdk-diff-dev*, che utilizza il context denominato *"dev"* all'interno del *cdk.json*, passato tramite il flag *-c* a *config*, come descritto in precedenza.

## 8.3 Rilascio

Per effettuare il rilascio dell'infrastruttura a seguito delle precedenti operazioni è possibile eseguire il deploy degli stack creati.
Il comando impiegato è ***cdk deploy***. In particolare nel progetto ho utilizzato *npm run cdk-deploy-stack* che consente di rilasciare uno stack specifico, specificando il nome dello stack con il flag *–stack*. È infatti consigliato rilasciare gli stack singolarmente oppure è possibile lasciare che AWS deduca tutte le dipendenze e rilasciare l'intera infrastruttura con *npm run cdk-deploy-dev-all*. Anche qui il parametro del contesto è passato tramite il flag *-c*.
Il deploy degli stack genererà le risorse in AWS visualizzabili nella console di AWS Cloudformation, come mostrato nell'esempio della figura 8.2.

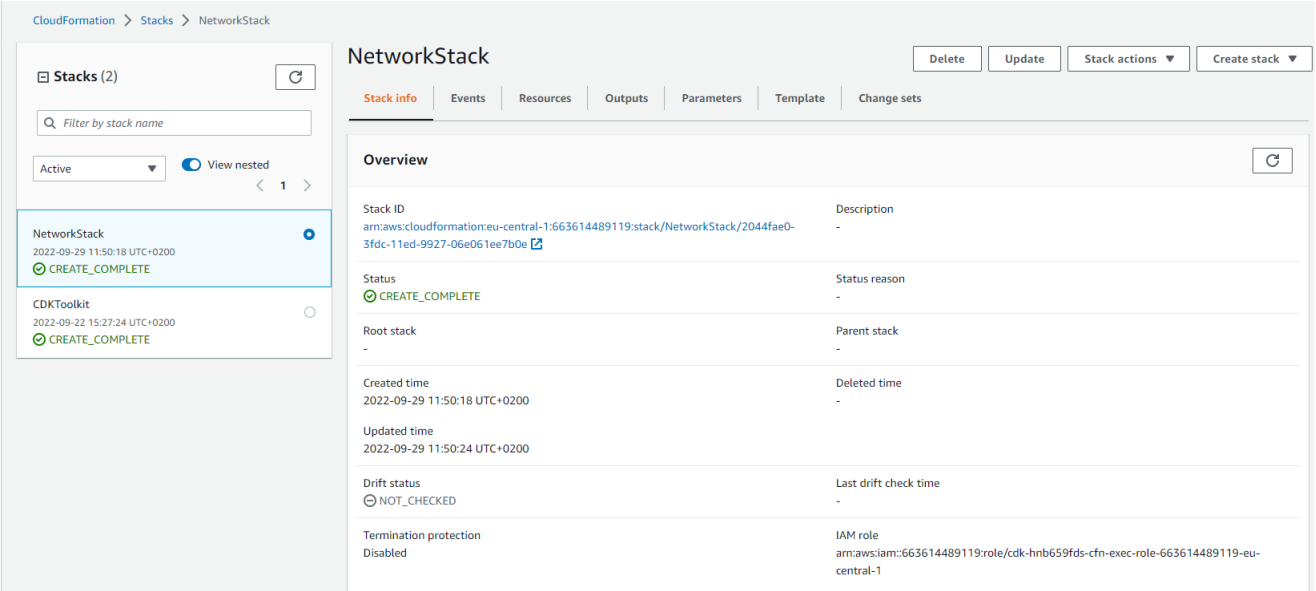


**Figura 8.2:** Esempio di stack distribuito in CloudFormation nella console AWS

## 8.4 Distruzione

Alla fine di ogni rilascio andato con successo è stato richiesto, per evitare costi inutili delle risorse rilasciate in AWS, di eliminare gli stack. Per effettuare questa operazione si utilizza il comando ***cdk destroy***.

In particolare ho utilizzato *npm run cdk-destroy-stack*, per eliminare i singoli stack tramite il flag *–stack*, oppure come per il deploy, tramite un unico comando ossia: *npm run cdk-destroy-dev-all*.

# Capitolo 9

# Conclusioni

In questa ultima sezione andrò ad esporre alcune considerazioni finali sulla tematica affrontata, gli studi svolti per affrontarla e i risultati prodotti dei progetti proposti per l'attività di tirocinio e, tramite un'analisi retrospettiva, fornirò una descrizione dell'esperienza svolta.

## 9.1 Risultati

La tematica che ho affrontato durante il periodo di tirocinio, ossia la programmabilità delle infrastrutture cloud tramite codice, mi ha permesso di approfondire le conoscenze acquisite durante i corsi di laurea ed ampliarle toccando con mano ambiti vicini a quelli di produzione. In particolare, grazie allo studio degli *Amazon Web Services*, e nello specifico i servizi offerti per lo sviluppo di infrastrutture, ho avuto la possibilità di entrare in contatto con le tecnologie Amazon in costante evoluzione di cui non ero a conoscenza e che al giorno d'oggi si rivelano fondamentali per quelle aziende che vogliono espandersi tramite le potenzialità offerte dal cloud computing.
I servizi che Amazon espone agli sviluppatori coprono moltissimi ambiti e i dettagli implementativi di ognuno garantiscono un elevato livello di controllo sul prodotto finale ma, proprio per questo motivo, necessitano anche di figure professionali in aziende come zero12 al fine di sfruttarne tutte le potenzialità.
Lo studio preliminare che ho svolto durante l'attività di tirocinio ha occupato buona parte delle ore di stage e si è rivelato per me importantissimo non solo per lo svolgimento del progetto ma anche per possedere un solido bagaglio culturale utile per entrare nel mondo del lavoro in questo ambito. Grazie ai video corsi fruibili sulla piattaforma Udemy sono riuscito fin dal primo giorno ad avere una veloce ma completa introduzione del mondo AWS che mi ha permesso di mettermi alla pari dei miei colleghi in poco tempo e in seguito di confrontarmi con loro con tutta facilità riguardo le tre infrastrutture e

il loro sviluppo.
Come descritto nelle sezioni precedenti, le infrastrutture su cui ho lavorato rappresentano una semplificazione rispetto all'ambiente vero e proprio di produzione ma costituiscono le principali soluzioni che vengono proposte, soprattutto per quanto riguarda le infrastrutture che offrono i propri servizi grazie al paradigma *serverless* e *container*. Infatti, oltre all'approfondimento dei servizi stessi di Amazon ho dovuto studiare concetti correlati ad essi.
Ad esempio per quanto le pratiche di Continuous Integration e Continuous Delivery come metodologie di sviluppo per impostare un servizio di pipeline, i concetti che stanno alla base di un prodotto software offerto tramite il paradigma *serverless* e quindi l'implementazione tramite servizi progettati in maniera specifica per soddisfare tali requisiti oppure servizi quali Docker e Kubernetes per applicare la forma containerizzata come modalità di distribuzione. Tutto questo si adatta in maniera efficace al paradigma del cloud computing mettendone in luce le potenzialità che offre.
Inoltre, come ho menzionato, non era richiesto lo sviluppo di applicazioni che potessero essere eseguite sulle infrastrutture create in quanto usciva dagli obiettivi dello stage e per via della differenza che distingue le due aree.

## 9.2 Resoconto

Lo stage svolto preso zero12 si è rivelato per me molto interessante e stimolante, sia per le tematiche trattate sia per l'ambiente di lavoro nel quale mi trovavo. Fin da subito l'accoglienza e la disponibilità da parte dei colleghi è stata un punto di forza che mi ha permesso di imparare e lavorare in maniera efficace al progetto di stage. Inoltre i momenti di pausa, di gioco o di discussione sono stati particolarmente utili anche per conoscere quei colleghi in ufficio non strettamente vicini al mio ambito e alla mia area.
Dato il tema affrontato, in azienda sono stato assegnato all'area infrastrutture e grazie, soprattutto alla guida di Alessandro Barcaro sotto la supervisione del tutor aziendale Stefano Dindo, nonchè all'aiuto di coloro con cui ero vicino, sono riuscito ad acquisire un insieme di conoscenze sia strettamente legate all'ambito stesso sia alla collaborazione in un team di lavoro, che ritengo valide per la mia futura carriera lavorativa.
Nonostante i prodotti effettivamente sviluppati differiscano da quanto pianificato dal progetto di stage, le modifiche sono state fatte in corso d'opera per via dell'elevato carico di lavoro e del sufficiente materiale rispetto al monte ore del tirocinio.
Infine, durante lo stage ho espresso una certa volontà di poter continuare la collaborazione con zero12 dopo il termine degli studi, per approfondire quanto già appreso.

# Glossario

**API** Il termine API, acronimo di Application Programming Interface (interfaccia di programmazione delle applicazioni), indica un insieme di definizioni e protocolli per la creazione e l'integrazione di applicazioni software. Permettono ai tuoi prodotti o servizi di comunicare con altri prodotti o servizi, anche se non sai come sono stati implementati, semplificando così lo sviluppo delle app e consentendo un netto risparmio di tempo e denaro . 6, 19, 30, 40, 42, 44, 51

**CDK** È un framework di sviluppo software open-source per definire l'infrastruttura cloud come un codice tramite i moderni linguaggi di programmazione e per distribuirla attraverso AWS CloudFormation. 20, 26, 30, 52

**composizione** È un principio di programmazione secondo cui le classi dovrebbero ottenere il comportamento polimorfico e il riutilizzo del codice mediante la composizione, ossia contenere altre classi che implementano la funzionalità desiderata, invece che attraverso l'ereditarietà, cioè tramite una o più sottoclassi. 20, 26, 27

**JSON** JSON è un formato di serializzazione basato su testo per lo scambio di dati, principalmente tra server e applicazione web. JSON è l'acronimo di JavaScript Object Notation, e utilizza l'estensione di file .json. Il formato JSON è basato su due tipi di strutture di dati: serie di coppie nome/valore - "name1":"value1", "name2":"value2" - e liste ordinate di valori, chiamate array - ["valore1", "valore2"]. Questi dati vengono letti da un parser JSON che li converte nel tipo di dati appropriato per il linguaggio di programmazione usato nel recupero dei dati. 9, 13, 18, 20, 30, 43

**network** Si intende comunemente un insieme di nodi interconnessi da canali di comunicazione per lo scambio di informazioni come dati e messaggi. 6, 10, 14, 30, 32, 48

**pipeline** È un insieme di processi e strumenti automatizzati che consente agli sviluppatori e ai professionisti delle operazioni di collaborare in modo coeso alla creazione e alla distribuzione di codice in un ambiente di produzione. Sebbene una pipeline possa differire a seconda dell'organizzazione, in genere include funzionalità di

automazione, continuous integration delle build, test di automazione, convalida e reporting . 32, 36, 48, 49, 60

**server** In informatica è un dispositivo fisico o sistema informatico di elaborazione e gestione del traffico di informazioni. Un server fornisce, a livello logico e fisico, un qualunque tipo di servizio ad altre componenti (tipicamente chiamate client) che ne fanno richiesta attraverso una rete di computer, all'interno di un sistema informatico o anche direttamente in locale su un computer. 5, 6, 10, 16, 21, 39–41, 43, 44, 49, 52, 53

**template** In informatica indica un documento o programma nel quale, come in un foglio semicompilato cartaceo, su una struttura generica o standard esistono spazi temporaneamente vuoti e riempibili con i valori voluti. 4, 9, 29, 52

**traffico** Il traffico Internet è il flusso di dati all'interno dell'intera Internet o in determinati collegamenti di rete delle sue reti costitutive. Le misurazioni del traffico comuni sono il volume totale, in unità di multipli del byte o come velocità di trasmissione in byte per determinate unità di tempo . 15, 16, 32–34, 36, 50

# Bibliografia


[1] *What is Infrastructure as Code:*
https://www.hpe.com/it/it/what-is/infrastructure-as-code.html

[2] *What is Cloud computing*
https://www.hpe.com/it/it/what-is/cloud-computing.html

[3] *Cloud computing types*
https://www.redhat.com/en/topics/cloud-computing/
public-cloud-vs-private-cloud-and-hybrid-cloud#overview

[4] *AWS Global Infrastrcucture*
https://aws.amazon.com/it/about-aws/global-infrastructure/

[5] *Console AWS* https://aws.amazon.com/it/console/

[6] *Amazon Documentation* https://docs.aws.amazon.com/

[7] *CDK API Reference v2 - Construct Library*
https://docs.aws.amazon.com/cdk/api/v2/

[8] *AWS CDK Developer Guide*
https://docs.aws.amazon.com/cdk/v2/guide/home.html

[9] *Node JS* https://nodejs.org/it/download/

[10] *Guida a TypeScript* https://www.html.it/guide/guida-typescript/

[11] *Visual Studio Code* https://code.visualstudio.com/download

[12] *What is API* https://aws.amazon.com/it/what-is/api/

[13] *Docker* https://aws.amazon.com/it/docker/

[14] *Kubernetes* https://kubernetes.io/docs/home/