Lezione 2

IMPORTANTE:
Può essere utile ricevere delle notifiche da AWS quanto si supera un certo limite di spesa. Le istruzioni su come configurare questa funzionalità di *CloudWatch* si trovano qui:
`https://docs.aws.amazon.com/it_it/AmazonCloudWatch/latest/monitoring/monitor_estimated_charges_with_cloudwatch.html`

AWS Command Line Interface

The AWS Command Line Interface (AWS CLI) is an open source tool that enables you to interact with AWS services using commands in your command-line shell. With minimal configuration, you can start using functionality equivalent to that provided by the browser-based AWS Management Console.

- **Installazione:**
  `https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-install.html`
- **Configurazione:** eseguire `aws configure` ed inserire:
  - `AWS Access Key ID` e `AWS Secret Access Key` (credenziali AWS)
  - regione di default da utilizzare (e.g., `us-west-2`)
  - formato di output di default (e.g., `json` o `text`)
  - Le istruzioni su come ottenere le access key sono qui:
    `https://docs.aws.amazon.com/it_it/cli/latest/userguide/cli-chap-configure.html`

Utilizzo di EC2 attraverso AWS CLI

Creiamo un security group da riga di comando e avviamo una istanza EC2.

Prendiamo nota dell'ID del VPC di default: `vpc-XXXXXXX`

```
$ aws ec2 create-security-group --group-name my-sg\
    --description "My security group" --vpc-id vpc-d5c125bc
```

Per visualizzare i dettagli del gruppo appena creato, usando il suo id:

```
$ aws ec2 describe-security-groups --group-ids <ID DEL GRUPPO>
```

Definiamo delle regole per il traffico in ingresso verso il gruppo. Accettiamo traffico TCP sulla porta 22 (SSH) e 80 (WEB) da tutti gli indirizzi:

```
$ aws ec2 authorize-security-group-ingress --group-id <ID DEL GRUPPO> \
    --protocol tcp --port 22 --cidr 0.0.0.0/0
$ aws ec2 authorize-security-group-ingress --group-id <ID DEL GRUPPO> \
    --protocol tcp --port 80 --cidr 0.0.0.0/0
$ aws ec2 describe-security-groups --group-ids <ID DEL GRUPPO>
```

Avviamo un'istanza in una delle subnet del VPC (e.g., `subnet-0ef1f676`):

```
$ aws ec2 run-instances --image-id <ID AMI> --count 1 --instance-type t2.nano \
        --key-name MyKeyPair --security-group-ids <ID GRUPPO> \
        --subnet-id subnet-0ef1f676
```

Possiamo associare un *tag* all'istanza:

```
$ aws ec2 create-tags --resources <ID ISTANZA> --tags Key=Name,Value=SDCC
```

Per visualizzare le istanze attive filtrando per tag e tipo:

```
$ aws ec2 describe-instances --filters "Name=tag:Name,Values=SDCC"
$ aws ec2 describe-instances --filters "Name=instance-type,Values=t2.nano"
```

Per terminare l'istanza:

```
$ aws ec2 terminate-instances --instance-ids <ID>
```

Riferimenti:
`https://docs.aws.amazon.com/it_it/cli/latest/userguide/cli-services-ec2-instances.html`

Automatizzazione della configurazione con Ansible

Ansible is an IT automation tool. It can configure systems, deploy software, and orchestrate more advanced IT tasks such as continuous deployments or zero downtime rolling updates.

Caratteristiche/concetti principali:

- Inventory
- *Agentless*: non serve installare alcun software aggiuntivo sulle macchine da configurare (unico requisito: Python 2.5+)
- Basato su una ampia collezione di moduli
- Idempotenza
- Playbook (YAML)

Vediamo un esempio d'uso di Ansible. Per tutti i dettagli e per approfondire i concetti fondamentali, fare riferimento alla documentazione ufficiale:
`https://docs.ansible.com/ansible/latest/index.html`

Deployment di una applicazione Flask su EC2 con Ansible

Come nella lezione precedente, avvieremo una applicazione web basata sul framework Flask usando una (o più) istanze EC2. Invece di effettuare la configurazione manualmente, definiamo un playbook Ansible.

L'applicazione che consideriamo 'e una *Photo Gallery*, che permette di visualizzare e pubblicare delle immagini.

Creiamo una cartella `photogallery_ansible`. Per prima cosa definiamo il nostro **inventory** file `hosts.ini`, inserendo come unico host l'istanza EC2 creata in precedenza. Per scenari pi'u complessi, 'e possibile aggiungere molteplici host (uno per riga) e organizzarli in gruppi (e.g., *web*, *database*, ...).

```
[web]
18.185.19.141 ansible_user='ec2-user' \
    ansible_ssh_private_key_file='/path/to/keypair-frankfurt.pem'
```

Per verificare che Ansible sia in grado di stabilire una connessione con i nodi nell'inventory:

```
$ ansible -i hosts.ini -m ping all
```

Per configurare la nostra applicazione dobbiamo:

- Eseguire l'upload dei file della applicazione (modulo **copy**)
- Installare le dipendenze della nostra applicazione (moduli **yum** e **pip**)
- Copiare il file relativo al servizio systemd per l'avvio del server nella

- cartella opportuna (modulo **copy**)
- Abilitare ed avviare il servizio (modulo **systemd**)

Creiamo il nostro **playbook**, scrivendo il file `deploy_gallery.yaml`:

```
---
- hosts: web
  vars:
    local_app_dir: "../photogallery"
    remote_home: "/home/ec2-user"
    remote_app_dir: "{{ remote_home}}/photogallery"

  tasks:
    - name: copy application directory
      copy:
        src: "{{ local_app_dir }}/"
        dest: "{{ remote_app_dir }}"

    - name: install python 3
      become: yes # we need root privileges
      yum:
        name: python3
        state: present

    - name: install required Python modules
      become: yes
      pip:
        name: flask
        executable: pip3

    - name: copy systemd unit file
      become: yes
      copy:
        src: "photogallery.service"
        dest: "/etc/systemd/system/"

    - name: enable and start systemd service
      become: yes
      systemd:
        daemon_reload: yes
        state: started
        name: "photogallery.service"
        enabled: yes
```

Eseguiamo il playbook con:

```
ansible-playbook -i hosts.ini deploy_gallery.yaml
```

Proviamo ad eseguirlo di nuovo: vediamo che Ansible non esegue nuovamente le operazioni, dato che la macchina si trova già nella configurazione richiesta.

Amazon S3: Simple Storage Service

- S3 è un servizio di object storage scalabile fornito da AWS
- Pricing: `https://aws.amazon.com/it/s3/pricing/`
- Basato su **bucket**
- Creiamo un bucket tramite l'interfaccia web
- Il nome di un bucket deve essere univoco all'interno di S3!
- Creiamo delle directory e carichiamo dei file
- Abbiamo il controllo su chi può accedere agli oggetti e al bucket: `https://docs.aws.amazon.com/it_it/AmazonS3/latest/dev/example-bucket-policies.html`

Se accessibile pubblicamente, un oggetto all'interno di un bucket può essere indirizzato in questo modo:

`https://NOMEBUCKET.s3.amazonaws.com/NOMEFILE`

Possiamo usare AWS CLI per interagire con S3, e.g.:

```
$ aws s3 ls
$ aws s3 ls s3://mybucket
$ aws s3 cp prova.txt s3://mybucket/
$ aws s3 ls s3://mybucket
$ aws s3 rm s3://mybucket/
```

Riferimento:
`https://docs.aws.amazon.com/cli/latest/reference/s3/`

AWS SDK per Python: boto3

Boto is the Amazon Web Services (AWS) SDK for Python. It enables Python developers to create, configure, and manage AWS services, such as EC2 and S3. Boto provides an easy to use, object-oriented API, as well as low-level access to AWS services.

Oltre alla CLI, AWS offre delle API per interagire con i servizi cloud all'interno di programmi pi'u complessi. Le API AWS sono disponibili per diversi linguaggi. La libreria che useremo per interagire con AWS usando Python si chiama **boto3**.

Documentazione, con istruzioni su come installare la libreria e le funzionalità offerte: `https://boto3.amazonaws.com/v1/documentation/api/latest/index.html`

Configurazione: se avete configurato AWS CLI sulla macchina, non serve fare altro.

Proviamo ad usare boto3 per accedere al nostro bucket su S3: `s3list.py`

Esercizio

Estendere l'applicazione *PhotoGallery* utilizzata durante la lezione implementando le seguenti funzionalità:

- l'applicazione visualizza le immagini presenti in un object storage nel cloud (e.g., un bucket S3);
- l'applicazione consente l'upload di nuove immagini;
- l'applicazione visualizza per ogni immagine la data in cui è stata pubblicata.

Effettuare il deployment dell'applicazione usando AWS, prestando attenzione alla scalabilità dell'infrastruttura utilizzata.

Suggerimento: per usare boto3 da una macchina EC2, dovete configurare le chiavi di accesso all'account AWS anche sull'istanza. Una possibilità, se avete configurato AWS CLI sul vostro pc, è eseguire l'upload del file `/home/NomeUtente/.aws/credentials` in `/home/ec2-user/.aws/`, oppure, se dovete usare la libreria in un processo che esegue come utente root, in `/root/.aws/`.

ATTENZIONE:
Non caricate le vostre chiavi di accesso AWS in alcun repository/servizio di storage!!

Metodi alternativi per configurare boto3:
`https://boto3.amazonaws.com/v1/documentation/api/latest/guide/configuration.html`