

La AWS CLI (Command Line Interface) - Premesse

La AWS CLI (versione 2) è un tool software open source

(<https://github.com/aws/aws-cli/tree/v2>), che fornisce comandi di livello shell/interprete dei comandi per interagire con i servizi AWS

- È scritta in Python su Boto, SDK Python per AWS; attraverso Boto, il codice Python di AWS CLI pilota la API AWS e interagisce così con il cloud AWS
- È funzionalmente equivalente alla console Web di AWS (con la CLI si può fare sul Cloud tutto ciò che si può fare con la console Web)
- Si può utilizzare su un cliente da shell Unix o da Powershell/interprete dei comandi Windows o, in remoto, collegandosi con *ssh* a un'istanza EC2 dotata di AWS CLI (p.es. distro Amazon Linux), oppure ancora via CloudShell AWS
- AWS CLI dà accesso “diretto” (1-1) alle API pubbliche di AWS, cioè per ogni funzione della API c'è un corrispondente comando CLI che la invoca
- AWS CLI fornisce anche comandi, di livello più alto, che semplificano l'accesso a servizi AWS complessi;
p.es. *aws s3...* fornisce un gruppo di comandi per la gestione di S3:

```
$ aws s3 cp myvideo.mp4 s3://mybucket/
```

Dietro l'apparente semplicità di questo comando, c'è un upload “multipart” e “robusto” di un grosso file (*myvideo.mp4*); effettuarlo a livello di operazioni (più elementari) di API, ovvero usando i comandi CLI “*aws s3api...*”, sarebbe ben più complicato.

- In ogni caso, AWS CLI permette di sviluppare script di shell, complessi a piacimento, con cui gestire risorse AWS

La API di AWS

La “vera” interfaccia di AWS – Amazon Web Services – verso il mondo esterno è una API HTTP REST JSON. Il “vero” modo, e più basilare (seppur macchinoso), di interagire con AWS è quindi di:

1. stabilire una connessione con un *endpoint* REST (ogni endpoint è una URL, specifica di una regione), inviando un'opportuna richiesta HTTP contenente un *payload* JSON che descrive l'operazione richiesta;
(il richiedente deve (in qualche modo) autenticarsi con AWS e firmare digitalmente la richiesta)
2. attendere e leggere la risposta HTTP, che potrebbe contenere il risultato dell'operazione, sempre in JSON.

Questo non è ovviamente il modo raccomandato, o più semplice, di accedere a AWS, ma potreste avere la curiosità di condurre qualche esperimento, v.

<https://blog.scottlowe.org/2020/04/10/using-postman-to-launch-ec2-instance-via-api-calls/> e <https://blog.scottlowe.org/2020/02/27/region-endpoint-match-in-aws-api-requests/>

La documentazione ufficiale AWS al riguardo:

<https://docs.aws.amazon.com/AWSEC2/latest/APIReference/making-api-requests.html>.

Accesso “programmatico” a AWS via API/SDK

Come detto, l’accesso “manuale” a AWS può avvenire via console Web o AWS CLI, mentre per l’accesso programmatico si può ricorrere a script di shell contenenti opportune invocazioni della AWS CLI.

L’alternativa all’impiego di tali script di shell è lo sviluppo di codice per linguaggi/ambienti dotati di API/SDK per l’accesso a AWS. P.es., anziché uno script Bash, uno script Python o una app Java, che, attraverso il relativo SDK (per Python o Java rispettivamente) acceda alla API di AWS e interagisca così col cloud AWS.

Anche la console web è in effetti un’interfaccia (Web) verso la API REST... Immaginiamo di avere avviato e poi terminato un’istanza:

<input type="checkbox"/>	Name ▲	Instance ID	Instance state ▼	Availability Zone ▼	VPC ID
<input checked="" type="checkbox"/>	–	i-079fa23d0bb8a1eee	Terminated 🔍	us-east-1a	vpc-4fa79129

Il servizio AWS *CloudTrail* fornisce un log delle chiamate API effettuate da un’utenza, comunque siano state generate (da CLI, console Web, codice via SDK o altro ancora):

CloudTrail > Event history						
Event history (2/5) Info						
Event history shows you the last 90 days of management events.						
Resource type ▼		Q AWS::EC2::Instance X		30m 1h 3h 12h Clear	Custom (15m)	< 1 > ⚙
<input checked="" type="checkbox"/>	Event name	Event time	User name	Event source	Resource type	Resource name
<input checked="" type="checkbox"/>	TerminateInstances	May 06, 2022, 08:30:...	root	ec2.amazonaws.com	AWS::EC2::Instance	i-079fa23d0bb8a1eee
<input checked="" type="checkbox"/>	RunInstances	May 06, 2022, 08:22:...	root	ec2.amazonaws.com	AWS::EC2::VPC, AWS::EC2::Ami, ...	vpc-4fa79129, ami-00...

Selezionando entrambi gli eventi, ne vedremo le caratteristiche; è relativamente semplice ricostruire il formato delle richieste alla API e risposte che hanno portato, in questo esempio, a creare e poi terminare l'istanza.

	Event 1 ✕	Event 2 ✕
Event name	TerminateInstances	RunInstances
Event ID	b1bdddd29-9d40-4337-8d21-4283e2c01a0e	3a678a43-c892-49a4-9632-3cf9076323b8
Event time	May 06, 2022, 08:30:17 (UTC+02:00)	May 06, 2022, 08:22:12 (UTC+02:00)
User name	root	root
AWS access key	ASIAT7AMEIYOZWIB23ZH	ASIAT7AMEIYOZWIB23ZH
Event source	ec2.amazonaws.com	ec2.amazonaws.com
Request ID	0e1f2955-ce23-43c8-b8b7-a90641ee24e6	6d083790-3841-491a-af33-4799d0672892
Error code	-	-
Source IP address	AWS Internal	AWS Internal
Read-only	false	false
AWS region	us-east-1	us-east-1
Event type	AwsApiCall	AwsApiCall

Elenco (con URL) di risorse per le principali alternative a AWS CLI per la shell di Unix:

- [AWS Tools for Windows PowerShell](#)
- [AWS SDK for Java](#)
- [AWS Toolkit for Eclipse](#) (supporta sviluppo di app Java con AWS SDK Java)
- [AWS Toolkit for VS Code](#) (estensione, supporta interazione con AWS via CLI, e da codice Java, Go, Python, node.js, .NET)
- [AWS SDK for .NET](#)
- [AWS Toolkit for Visual Studio](#) (supporta sviluppo di app .NET con AWS SDK .NET)
- [AWS SDK for JavaScript](#)
- [AWS SDK for Ruby](#)
- [AWS SDK for Python \(Boto\)](#)
- [AWS SDK for PHP](#)
- [AWS SDK for Go](#)

In generale, sulle API REST si veda:

<https://www.smashingmagazine.com/2018/01/understanding-using-rest-api/>

Convenzioni negli esempi AWS CLI

In *rosso corsivo* le parti variabili o *parametri* di un comando, e l'input fornito interattivamente da tastiera. Per esempio:

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
```

L'output dei comandi, costituito da metadati AWS, è per default in formato JSON. Per esempio:

```
$ aws ec2 create-security-group --group-name my-sg --description "My security group"
{
  "GroupId": "sg-903004f8"
}
```

AWS CLI: disponibilità

Installazione secondo la documentazione AWS

Per quanto AWS CLI sia un'applicazione Python, non conviene cercare di installarla collegandola "a mano" al Python di sistema, per non incappare nella gestione manuale delle complessità del versioning di Python e dei suoi file di supporto (librerie, etc.).

L'approccio raccomandato da AWS, per l'installazione, è descritto qui:

<https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html>. Le opzioni migliori sono:

1. accedere alla AWS CLI versione 2 dal browser, utilizzando AWS CloudShell, oppure
2. (Consigliato) installare (o aggiornare) AWS CLI sul proprio cliente desktop, v.
https://docs.aws.amazon.com/it_it/cli/latest/userguide/getting-started-install.html.

Seguendo (2), verrà installato tutto il bundle necessario (compreso Python). Una volta installata la CLI, si può determinare la versione disponibile con:

```
$ aws --version  
aws-cli/2.5.3 Python/3.9.12 Darwin/21.4.0 source/x86_64 prompt/off
```

mentre l'ultima versione disponibile è rilevabile qui: <https://github.com/aws/aws-cli/tree/v2>

Installazione di AWS CLI come pacchetto

In alternativa, alcune distribuzioni Linux (p.es. Debian/Ubuntu e Arch) prevedono pacchetti, aggiornati regolarmente, per AWS CLI. Si vedano, p.es., <https://packages.debian.org/search?keywords=awscli> e <https://www.archlinux.org/packages/community/any/aws-cli/>.

In questo caso, la gestione di eventuali dipendenze (Python e sue librerie) è automatizzata e delegata al sistema pacchettizzato (tipo *apt*, *yum* etc.). **Tuttavia, almeno per Ubuntu fino alla LTS 22.04 (maggio 2024), la versione di CLI installata è la 1, decisamente non più adeguata.**

Per Mac, il gestore di pacchetti *brew* consente di installare la "formula" *awscli*, che installa la CLI v. 2.

AWS CLI da un'istanza EC2

AWS CLI si può usare anche operando dalla shell di un'istanza EC2. AWS CLI è pre-installata su Amazon Linux, si può installare sugli altri tipi di immagine. In tal modo si eviteranno le problematiche di installazione/aggiornamento della CLI (con Python) sul proprio PC. Ovviamente, ci si dovranno sobbarcare i costi dell'uso dell'istanza.

Dall'istanza si può poi usare `aws configure` per configurare la AWS CLI, anche se la regione sarà pre-configurata. Come scelta per la regione di default, AWS CLI propone la stessa dell'istanza stessa (AWS CLI la legge direttamente dai metadati dell'istanza):


```
$ aws configure
AWS Access Key ID [None]: ENTER # salta
AWS Secret Access Key [None]: ENTER # skip
Default region name [None]: us-west-2 # valore proposto da AWS CLI
Default output format [None]: json
```

AWS CLI da CloudShell


CloudShell è un utilissimo servizio che mette a disposizione una shell all'interno del vostro browser. "Dietro" CloudShell vi è un container, ma, ai fini pratici, l'utente può immaginare sia un'istanza, però sempre disponibile e senza i costi associati all'istanza! I principali strumenti di sviluppo sono pre-installati.

Welcome to AWS CloudShell ×


AWS CloudShell is a browser-based shell that gives you command-line access to your AWS resources in the selected AWS Region. AWS CloudShell comes pre-installed with popular tools for resource management and creation. You have the same credentials as you use to log in to the console. [Learn more](#)



Pre-installed tools
AWS CLI, Python, Node.js and more



Storage included
1 GB of storage free per AWS Region



Saved files and settings
Files saved in your home directory are available in future sessions for the same AWS Region

Altri strumenti si possono installare con *yum* (la distribuzione Linux è quella *in-house* di AWS).

← → ↺

aws Services [Option+S]

N. Virginia Giuseppe Pappalardo

CloudShell Actions

us-east-1

```
[cloudshell-user@ip-10-140-98-113 ~]$ head -3 /etc/os-release
NAME="Amazon Linux"
VERSION="2023"
ID="amzn"
[cloudshell-user@ip-10-140-98-113 ~]$ sudo yum install php
Last metadata expiration check: 0:29:50 ago on Wed 17 Apr 2024 06:07:03 AM UTC.
Dependencies resolved.
=====
Package                Architecture  Version                                Repository      Size
=====
Installing:
php8.2                  x86_64        8.2.15-1.amzn2023.0.2                 amazonlinux     11 k
Installing dependencies:
apr                     x86_64        1.7.2-2.amzn2023.0.2                 amazonlinux     129 k
apr-util                x86_64        1.6.3-1.amzn2023.0.1                 amazonlinux     98 k
generic-logos-httpd    noarch       18.0.0-12.amzn2023.0.3                amazonlinux     19 k
httpd-core              x86_64        2.4.58-1.amzn2023                     amazonlinux     1.4 M
httpd-filesystem        noarch       2.4.58-1.amzn2023                     amazonlinux     14 k
httpd-tools             x86_64        2.4.58-1.amzn2023                     amazonlinux     81 k
```

Come vedremo, CloudShell è pre-autorizzata (senza chiavi) a eseguire la AWS CLI.

Configurare AWS CLI

AWS CLI deve essere configurata, tra l'altro, con le credenziali di sicurezza e una regione AWS di default.

AWS CLI comunica con i servizi/API del cloud AWS aprendo una connessione SSL per ogni richiesta. Ogni richiesta viene firmata digitalmente e la firma include un timestamp. AWS rifiuta richieste firmate con un timestamp (eccessivamente) disallineato rispetto al clock effettivo mantenuto da AWS.

È quindi essenziale che data/ora del cliente siano corrette.

Ausili

Completamento automatico (per Bash)

Probabilmente è già configurato e installato insieme alla CLI. Verificarlo con:

```
$ complete -p aws
complete -C 'aws_completer' aws
```

il che assicura il completamento automatico:

```
$ aws # qui battere il tasto Tab
Display all 308 possibilities? (y or n)
```

Qui lo si è visto in opera per i comandi di primo livello, ma funziona anche per il secondo livello (le “azioni”) e le opzioni (introdotte da -).

Se il completamento automatico funziona, potete saltare il resto di questa sezione. Se no, notare che, insieme a `aws`, nella cartella prevista:

```
$ which aws
/usr/local/bin/aws
```

è stato installato l'eseguibile `aws_completer`:

```
$ ls $(dirname $(which aws))/aws*
/usr/local/bin/aws  /usr/local/bin/aws_completer
```

Eseguire allora:

```
$ complete -C '/usr/local/bin/aws_completer' aws
```

Così si è installato `~/local/bin/aws_completer` come programma di completamento dei comandi della shell Bash per `aws`

Per rendere ciò permanente (nonché per completare il *PATH* in permanenza), scrivere:

```
$ cat >> ~/.bash_profile <<EOF
complete -C '~/local/bin/aws_completer' aws
export PATH=~/local/bin:$PATH
EOF
```

Auto-prompt semi-grafico

Un altro potente ausilio (derivato da iPython) si ottiene con

```
$ export AWS_CLI_AUTO_PROMPT=on
```

Dopodiché, se si tenta di eseguire un comando `aws` incompleto, verrà mostrata una schermata di aiuto e suggerimento:

```
gpp@AWS $ aws ec2 describe-i # premere Enter
> aws ec2 describe-i
describe-iam-instance-profile-associations
describe-id-format
describe-identity-id-format
describe-image-attribute
describe-images
describe-import-image-tasks
describe-import-snapshot-tasks
describe-instance-attribute
describe-instance-credit-specifications
describe-instance-event-notification-attributes
describe-instance-event-windows
describe-instance-status
```

[ENTER] Autocomplete Choice/Execute Command [F1] Show Shortkey Help [F2] Focus on next panel [F3] Hide/Show Docs [F5] Hide/Show Outpu

Maggiori dettagli sull'auto-prompting: <https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-parameters-prompting.html>

Configurazione veloce

Basta usare il comando `aws configure` (v. oltre come ottenere la Access Key (ID e parte segreta)):

```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

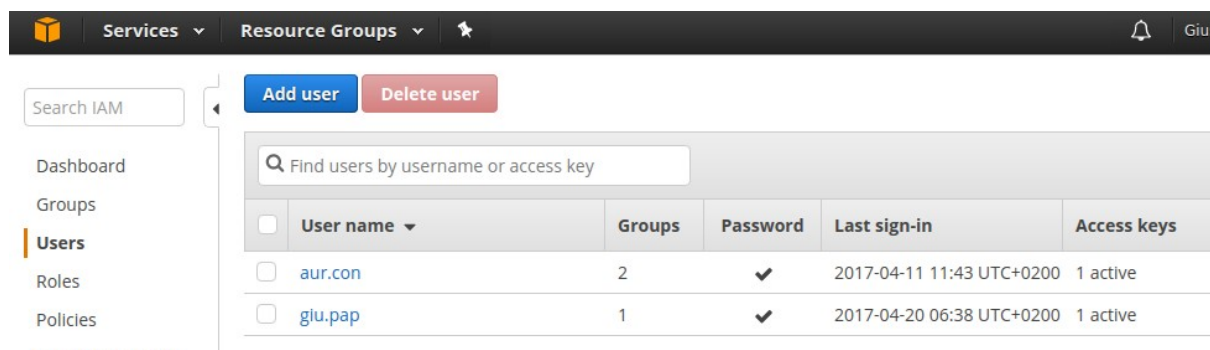
Per aggiornare le configurazioni, si lancia di nuovo:

```
$ aws configure
```

Accesso

Ottenere (creare) access key ID e secret access key: via IAM

Elenchiamo gli utenti IAM:



The screenshot shows the AWS IAM console interface. On the left is a navigation menu with 'Users' selected. The main area displays a table of IAM users. At the top of the table is a search bar labeled 'Find users by username or access key'. The table has columns for 'User name', 'Groups', 'Password', 'Last sign-in', and 'Access keys'. Two users are listed: 'aur.con' and 'giu.pap'. The 'giu.pap' user is highlighted, indicating it is the one being referenced in the text.

<input type="checkbox"/>	User name	Groups	Password	Last sign-in	Access keys
<input type="checkbox"/>	aur.con	2	✓	2017-04-11 11:43 UTC+0200	1 active
<input type="checkbox"/>	giu.pap	1	✓	2017-04-20 06:38 UTC+0200	1 active

NB: non usare mai la chiave di accesso dell'utente `root`, bensì quella di un utente con meno permessi, in questo esempio è `giu.pap`

Summary

User ARN: arn:aws:iam::272755869213:user/giu.pap
 Path: /
 Creation time: 2017-04-11 05:02 UTC+0200

Permissions Groups (1) **Security credentials** Access Advisor

Sign-in credentials

Console password: Enabled [Manage password](#)
 Console login link: https://272755869213.signin.aws.amazon.com/console
 Last login: 2017-04-20 06:38 UTC+0200
 Assigned MFA device: arn:aws:iam::272755869213:mfa/giu.pap
 Signing certificates: None

Access keys

Use access keys to make secure REST or HTTP Query protocol requests to AWS service APIs. For your protection, you should never share your secret keys with anyone. As a best practice, we recommend frequent key rotation. [Learn more](#)

[Create access key](#)

Access key ID	Created	Last used	Status
AKIAJC23YEZRORDLM6PA	2017-04-11 06:20 UTC+0200	N/A	Active Make inactive

N.B.: la parte segreta di una chiave esistente (già creata) doveva essere stata memorizzata localmente in precedenza al momento della creazione (tipicamente in un file .csv) e ora andrà recuperata (da quel file), per esempio:

```
$ csvlook accessKeys_2018.csv
| Access key ID | Secret access key |
| ----- | ----- |
| AKIAIGYK4AABAEPPQ7JA | y7v2qWJJl uvH9aW6qeuuBrR4/nZzrBxA3Bo00oZH |
```

Utenza AWS Academy

Se siete utenti di AWS Academy, anziché di AWS “commerciale”, IAM non consente di creare utenze. La chiave per l’utente può essere copiata da ~/.aws/credentials o visualizzata dal bottone *AWS Details – Show*.

Accesso senza chiave

Di recente, AWS sconsiglia l’uso di chiavi permanenti, come quelle appena discusse, per ovvie ragioni di sicurezza.

Ovviamente, inoltre, sarebbe una pessima idea includere la chiave nel codice eseguibile (o sorgente) che verrà distribuito (si pensi a una app che sfrutti i servizi AWS, p.es. come *back-end*).

Ciò che serve è consentire al codice un accesso temporaneo (e selettivo) a AWS. Si veda: <https://docs.aws.amazon.com/IAM/latest/UserGuide/security-creds.html> e, specificamente, <https://docs.aws.amazon.com/IAM/latest/UserGuide/security-creds.html#sec-alternatives-to-long-term-access-keys>

Un’alternativa è usare il servizio *CloudShell*, che fornisce, all’interno del vostro browser, una shell pre-autorizzata senza chiavi a eseguire la AWS CLI.


```
CloudShell
us-east-1

[cloudshell-user@ip-10-132-43-168 ~]$ aws ec2 describe-instances --query Reservations[0].Instances[0].PrivateIpAddress
"10.2.0.21"
[cloudshell-user@ip-10-132-43-168 ~]$ aws ec2 describe-instances --query Reservations[1].Instances[0].PrivateIpAddress
"10.6.0.134"
[cloudshell-user@ip-10-132-43-168 ~]$ ls -a .aws
ls: cannot access '.aws': No such file or directory
[cloudshell-user@ip-10-132-43-168 ~]$
```

Profili

Configurazione con profili multipli

Si possono avere anche profili multipli:

```
$ aws configure --profile user2
AWS Access Key ID [None]: AKIAI44QH8DHBEXAMPLE
AWS Secret Access Key [None]: je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
Default region name [None]: us-east-1
Default output format [None]: text
```

Dove risiedono le configurazioni

AWS CLI le cerca in questi posti (se ci sono), in quest'ordine:

1. **Command Line Options** – dove si possono specificare *region*, *output format* e *profile*
2. **Environment Variables** – quali `AWS_ACCESS_KEY_ID`, `AWS_ACCESS_KEY`, etc.
3. **file delle credenziali AWS** – `~/.aws/credentials` su Linux e macOS o `C:\Users\USERNAME\.aws\credentials` su Windows. Può contenere molteplici profili con nome in aggiunta al profilo di default.
4. **file di configurazione di AWS CLI** – `~/.aws/config` su Linux e macOS o `C:\Users\USERNAME\.aws\config` su Windows, può contenere un profilo di default, profili multipli con nome e, per ciascuno, specifici **parametri** di configurazione
5. **credenziali per un'istanza EC2 con AWS** – consegnati a EC2 attraverso il metadata service.

I file *config* e *credentials*

Si trovano in una cartella invisibile (Unix):

```
$ ls ~/.aws
cli/      config      credentials  shell/
```

Il file `config` contiene di norma i setting diversi dalle credenziali. Se ne può usare anche un altro, specificato con la variabile d'ambiente `AWS_CONFIG_FILE`. Il file `config` può anche contenere delle credenziali, ma si potrebbero avere dei warning e, comunque, quelle nel file `credentials` hanno la precedenza.

Vediamo i formati di questi file:

`~/.aws/config`

```
[default]
region=us-west-2
```

```
output=json
```

~/.aws/credentials

```
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxrFcYEXAMPLEKEY
aws_session_token=... # utilizzato se si usano credenziali di sicurezza temporanee
```

Profili con nome

Profili multipli, individuati con un nome, sono memorizzati nei file *config* e *credentials* e definiti con il comando:

```
$ aws configure --profile nome-attribuito-al-profilo
```

In questo caso, il file *credentials* conterrà due profili, quello di *default* e quello del profilo appena creato, nell'esempio *user2*:

```
$ cat ~/.aws/credentials
[default]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxrFcYEXAMPLEKEY

[user2]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

Anche il file *config* avrà due profili:

```
$ cat ~/.aws/config
[default]
region=us-west-2
output=json

[profile user2]
region=us-east-1
output=text
```

NB: nei due file si usano formati diversi per l'intestazione del profilo con nome (nell'esempio *user2*).

Specificare un profilo per i comandi della AWS CLI

I comandi accettano l'opzione `--profile`. Ad esempio:

```
$ aws ec2 describe-instances --profile user2
```

Oppure, si può cambiare il profilo implicito usato dai comandi con una variabile d'ambiente:

```
$ export AWS_DEFAULT_PROFILE=user2
```

Variabili d'ambiente per AWS CLI

Definiscono parametri che prendono la precedenza su quelli dei file di configurazione.

AWS_ACCESS_KEY_ID

AWS_SECRET_ACCESS_KEY

AWS_SESSION_TOKEN utilizzato se si impiegano credenziali di sicurezza temporanee.

AWS_DEFAULT_REGION

AWS_DEFAULT_PROFILE – nome di un profilo memorizzato nei file *credential* o *config*, o default per usare il profilo di default

AWS_CONFIG_FILE

AWS_CLI_AUTO_PROMPT

Per esempio:

```
$ export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
$ export AWS_DEFAULT_REGION=us-west-2
```

Alcune opzioni dei comandi AWS CLI

Permettono di cambiare setting di configurazione, ma non credenziali, per la singola invocazione di un comando:

--profile

--region

--output

--endpoint-url (di norma non impiegato perché la regione di riferimento ha un endpoint-url di default)

--cli-auto-prompt

Per esempio: se non si ricorda in che regione si trovano le nostre istanze EC2:

```
$ aws ec2 describe-instances --output table --region us-east-1
-----
|DescribeInstances|
+-----+
$ aws ec2 describe-instances --output table --region us-west-1
-----
|DescribeInstances|
+-----+
$ aws ec2 describe-instances --output table --region us-west-2
-----
|                                     DescribeInstances                                     |
+-----+-----+-----+-----+-----+-----+
||                                     Reservations                                     ||
+-----+-----+-----+-----+-----+-----+
|| OwnerId           | 012345678901 |
|| ReservationId     | r-abcdefgh  |
+-----+-----+-----+-----+-----+-----+
||                                     Instances                                     ||
+-----+-----+-----+-----+-----+-----+
|| AmiLaunchIndex    | 0           |
|| Architecture      | x86_64      |
|| ...               |             |
```

Assumere un ruolo nella AWS CLI

Si può configurare AWS CLI per usare un ruolo creando un profilo con nome uguale al ruolo nel file `~/.aws/config`. In questo profilo-ruolo, si indicherà anche quale altro profilo vero e proprio (`source_profile`) può assumere il ruolo.

In questo esempio, viene introdotto il profilo-ruolo *marketingadmin* (che è un ruolo, definito in qualche modo, via Web console o con la stessa AWS CLI) e si autorizza il profilo di default ad assumere il ruolo:

```
[profile marketingadmin]  
role_arn = arn:aws:iam::123456789012:role/marketingadmin  
source_profile = default
```

Si presuppone che il `source_profile default` abbia chiave corrispondente a un utente IAM (p.es. *giu.pap.18*) cui è permesso assumere il ruolo *marketingadmin*, chiamando il servizio AWS STS `sts:assume-role`. Dal canto suo, il ruolo deve avere una relazione di trust con l'utente, affinché questo possa assumerlo.

Ecco un esempio di relazione di trust che, se definita (NdGP: attribuita? Collegata?) per un ruolo come *marketingadmin* permette che esso sia assunto da un utente IAM *giu.pap.18* (inserire ARN corretto):

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Sid": "",  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::123456789012:user/giu.pap.18"  
      },  
      "Action": "sts:AssumeRole"  
    }  
  ]  
}
```

Inoltre, affinché *giu.pap.18* possa effettivamente assumere il ruolo *marketingadmin*, deve godere di una policy IAM come la seguente (inserire ARN corretto):

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Action": "sts:AssumeRole",  
      "Resource": "arn:aws:iam::123456789012:role/marketingadmin"  
    }  
  ]  
}
```

A questo punto, usando il profilo *marketingadmin*, l'utente potrà lanciare comandi AWS CLI come:

```
$ aws s3 ls --profile marketingadmin  
2018-03-28 10:23:11 dmi.sd2.bucket1
```

Se poi volesse assumere il ruolo *marketingadmin* come default per i successivi comandi AWS CLI:

```
$ export AWS_DEFAULT_PROFILE=marketingadmin
```

Ruoli e Autenticazione Multifattoriale

Per maggiore sicurezza, si può imporre che, quando l'utente effettua chiamate usando il ruolo, debba fornire una one-time password generata da un dispositivo. A questo scopo, la relazione di trust del ruolo verso l'utente può essere "condizionata", richiedendo appunto la "multifactor authentication":

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": { "AWS": "arn:aws:iam::123456789012:user/jonsmith" },
      "Action": "sts:AssumeRole",
      "Condition": { "Bool": { "aws:MultiFactorAuthPresent": true } }
    }
  ]
}
```

Ora, sulla macchina da cui si userà AWS CLI, si specifica il token che *jonsmith* userà:

```
[profile marketingadmin]
role_arn = arn:aws:iam::123456789012:role/marketingadmin
source_profile = default
mfa_serial = arn:aws:iam::123456789012:mfa/jonsmith
```

Ruoli Cross Account

Come si ricorderà, un ruolo IAM per un account può essere configurato per essere assunto da utenti di altri account, attraverso un meccanismo di *external_id*. Immaginiamo che ciò sia accaduto per il ruolo *xaccount*, per il quale è anche richiesta la MFA nella relazione di trust tra il ruolo e l'utente *jonsmith* (come nell'esempio precedente).

Nel profilo del ruolo, sul cliente AWS CLI si aggiungerà allora un parametro *external_id*:

```
[profile crossaccountrole]
role_arn = arn:aws:iam::234567890123:role/xaccount
source_profile = default
mfa_serial = arn:aws:iam::123456789012:mfa/jonsmith
external_id = 123456
```

Esempio: AWS CLI e AWS EC2

Creazione del security group

In previsione della creazione da CLI di istanze EC2, definiamo (sempre da CLI) un security group:

```
$ aws ec2 create-security-group --group-name devenv-sg --description "security group for development environment in EC2"
```

An error occurred (UnauthorizedOperation) when calling the CreateSecurityGroup operation: You are not authorized to perform this operation.

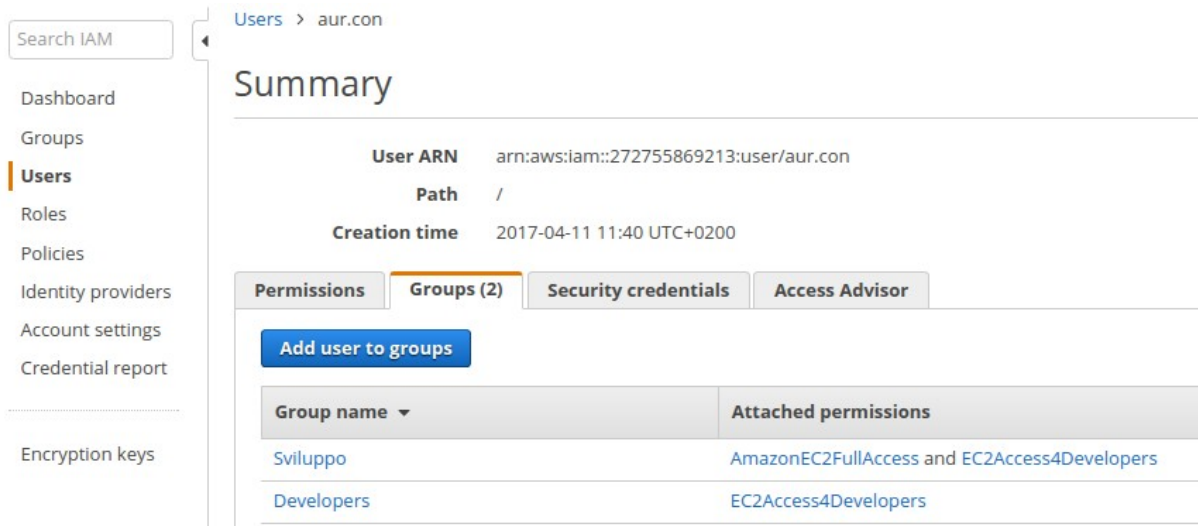
Problema: l'utente del profilo di default (chiunque egli sia) non ha il permesso di creare security group, ma c'è un altro utente con cui provare:

```
$ cat ~/.aws/config
[default]
region = us-east-1
[profile aur.con]
region = eu-west-1
```

Lo proviamo:

```
$ aws --profile=aur.con ec2 create-security-group --group-name devenv-sg --description "security group for development environment in EC2"
{
  "GroupId": "sg-da6d7aa3"
}
```

Spiegazione: dalla console IAM si vede che *aur.con* ha addirittura i permessi *AmazonEC2FullAccess*



The screenshot shows the AWS IAM console interface. On the left is a navigation menu with options like Dashboard, Groups, Users, Roles, Policies, etc. The main area displays the 'Summary' for the user 'aur.con'. It shows the User ARN as 'arn:aws:iam::272755869213:user:aur.con', the Path as '/', and the Creation time as '2017-04-11 11:40 UTC+0200'. Below this, there are tabs for 'Permissions', 'Groups (2)', 'Security credentials', and 'Access Advisor'. The 'Groups (2)' tab is active, showing a table of groups the user belongs to. The table has two columns: 'Group name' and 'Attached permissions'. The first row shows the 'Sviluppo' group with permissions 'AmazonEC2FullAccess and EC2Access4Developers'. The second row shows the 'Developers' group with permission 'EC2Access4Developers'.

Group name	Attached permissions
Sviluppo	AmazonEC2FullAccess and EC2Access4Developers
Developers	EC2Access4Developers

Come si vede, l'utente *aur.con* appartiene al gruppo *Sviluppo* che conferisce il permesso predefinito *AmazonEC2FullAccess*. Conviene quindi usarlo come profilo di default per i prossimi comandi:

```
$ export AWS_DEFAULT_PROFILE="aur.con"
```

Ora inseriamo nel nuovo security group *devenv-sg* una regola inbound che permetta traffico *ssh* in ingresso:

```
$ aws ec2 authorize-security-group-ingress --group-name devenv-sg --protocol tcp --port 22 --cidr 0.0.0.0/0
```

Per maggiore sicurezza, in alternativa, si potrebbe restringere l'accesso alla classe B di UniCT:

```
$ aws ec2 authorize-security-group-ingress --group-name devenv-sg --protocol tcp --port 22 --cidr 151.97.0.0/16
```

cancellando la regola troppo "aperta":

```
$ aws ec2 revoke-security-group-ingress --group-name devenv-sg --protocol tcp --port 22 --cidr 0.0.0.0/0
```

Verifichiamo l'effetto, sul security group `devenv-sg`, sempre con un comando AWS CLI:

```
$ aws ec2 describe-security-groups --group-name devenv-sg
...
    {
      "OwnerId": "272755869213",
      "GroupName": "devenv-sg",
      "GroupId": "sg-da6d7aa3",
      "Description": "security group for development environment in EC2",
      "IpPermissions": [
        {
          "IpProtocol": "tcp",
          "FromPort": 22,
          "ToPort": 22,
          "UserIdGroupPairs": [],
          "IpRanges": [
            {
              "CidrIp": "151.97.0.0/16"
            }
          ],
          "Ipv6Ranges": [],
          "PrefixListIds": []
        }
      ],
      "IpPermissionsEgress": [
      ]
      "VpcId": "vpc-f07abf97"
    },
    ...
```

Creazione (*launch*) dell'istanza

Prima di tutto, dotiamoci di una coppia di chiavi:

```
$ aws ec2 create-key-pair --key-name devenv-key --query 'KeyMaterial' --output text > devenv-key.pem
$ ls -l devenv-key.pem
-rw-r--r-- 1 gp gp 1675  8 giu 08.21 devenv-key.pem
```

Il file `devenv-key.pem` deve essere leggibile solo dall'*owner*:

```
$ chmod go-rw devenv-key.pem
$ ls -l devenv-key.pem
-rw----- 1 gp gp 1675  8 giu 08.21 devenv-key.pem
```

A questo punto, per lanciare l'istanza, occorre il comando

```
$ aws ec2 run-instances \
  --image-id ami-09d56f8956ab235b3 \      # vedi figura sotto, individua S.O. (Ubuntu) e regione (us-east-1)
  --security-group-ids sg-da6d7aa3 \    # è il security group creato in precedenza
  --count 1 \                             # n. di istanze (il comando è run-instances)
  --instance-type t2.micro \              # modello di memoria / ambiente di esecuzione, t2.micro è nel Free Tier
  --key-name devenv-key \                # è la coppia di chiavi generata prima
  --query 'Instances[0].InstanceId' \   # dall'output estraiamo l'id dell'istanza creata
  "i-07462a3fca71b342e"                  # questo output è l'instance id dell'istanza creata
```


La seconda figura sotto mostra (nei riquadri rossi) alcuni *image-id* tratti dalle schermata iniziale della console web *Launch instances*:

▼ Application and OS Images (Amazon Machine Image) Info

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. Search or Browse for AMIs if you don't see what you are looking for below

Search our full catalog including 1000s of application and OS images

Recents Quick Start

Amazon Linux macOS Ubuntu Windows Red Hat SUSE L

Browse more AMIs
Including AMIs from AWS, Marketplace and the Community

Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Selected AMI: (ami-051f8a213df8bc089) (Quickstart AMIs)

Search: Ubuntu

Quickstart AMIs (9) Commonly used AMIs | My AMIs (0) Created by me | AWS Marketplace AMIs (2404) AWS & trusted third-party AMIs | Community AMIs (500) Published by anyone

Refine results

Clear all filters

Free tier only info

OS category

All Linux/Unix

All Windows

Architecture

64-bit (Arm)

32-bit (x86)

64-bit (x86)

64-bit (Mac)

64-bit (Mac-Arm)

Ubuntu (9 filtered, 9 unfiltered)

ubuntu Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
ami-080e1f13689e07408 (64-bit (x86)) / ami-0a55ba1c20b74fc30 (64-bit (Arm))
Select
Free tier eligible
Verified provider
Ubuntu Server 22.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).
Platform: ubuntu Root device type: ebs Virtualization: hvm ENA enabled: Yes
64-bit (x86)
64-bit (Arm)

ubuntu Ubuntu Server 20.04 LTS (HVM), SSD Volume Type
ami-0cd59ecaf368e5ccf (64-bit (x86)) / ami-05f45f2e962205865 (64-bit (Arm))
Select
Free tier eligible
Verified provider
Ubuntu Server 20.04 LTS (HVM), EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).
Platform: ubuntu Root device type: ebs Virtualization: hvm ENA enabled: Yes
64-bit (x86)
64-bit (Arm)

ubuntu Ubuntu Server 20.04 LTS (HVM) with SQL Server 2022 Standard
ami-032346ab877c418af (64-bit (x86))
Select
Verified provider
Microsoft SQL Server 2022 Standard edition on Ubuntu Server 20.04 LTS.
Platform: ubuntu Root device type: ebs Virtualization: hvm ENA enabled: Yes
64-bit (x86)

Ogni AMI image-id è specifico della regione (qui us-east-1), del Sistema Operativo, dei tipi di Root device e virtualizzazione e dell'architettura (qui x86, 64 bit).

Vediamo ora l'IP pubblico dell'istanza creata:

```
$ aws ec2 describe-instances --instance-ids "i-07462a3fca71b342e" --query 'Reservations[0].Instances[0].PublicIpAddress'
"54.171.227.237"
```

Infine, colleghiamoci via ssh con l'istanza:

```
$ ssh -i devenv-key.pem ubuntu@54.171.227.237
```

Tutto ottenuto senza mai lasciare la shell!

Le immagini Ubuntu per il cloud

Canonical mette a disposizione uno strumento Web per scaricare le immagini pre-generate delle varie versioni di Ubuntu: <https://cloud-images.ubuntu.com/locator/>:

Search:

Cloud	Zone	Name	Version	Arch	Instance Type	Release	
Amazon AWS	us-east-1	jammy	22.04	amd64	hvm-ssd	20240326	ami-053053586808c3e70
Amazon AWS	us-east-2	jammy	22.04	amd64	hvm-ssd	20240326	ami-068cf3d51efeb20d6
Amazon AWS	us-east-1	jammy	22.04	amd64	hvm-ssd	20240319	ami-0e21465cede02fd1e
Amazon AWS	us-east-2	jammy	22.04	amd64	hvm-ssd	20240319	ami-065d315e052507855
Amazon AWS	us-east-1	jammy	22.04	amd64	hvm-ssd	20240301	ami-080e1f13689e07408
Amazon AWS	us-east-2	jammy	22.04	amd64	hvm-ssd	20240301	ami-0b8b44ec9a8f90422

Showing 1 to 6 of 6 entries (filtered from 75,991 total entries)

Come si vede, lo strumento fornisce anche gli *ami-id* per le immagini selezionate (con la search box e i menu *select* di scelta).

Elencare gli image id dalla CLI

In effetti, anche l'AMI image-id può essere ricavato con la CLI piuttosto che dalla console Web:

```
$ aws ec2 describe-images
```

... non fatelo! l'output tarda perché è molto lungo (nel 2024 circa 80.000 immagini!). L'output è in effetti un elenco *Images[]*, p.es.:

```
$ aws ec2 describe-images --query 'Images[0]'
```

...

[Molto tempo dopo]

```
$ aws ec2 describe-images --filters "Name=description,Values=*Ubuntu*22.04*" ...
```

Filtriamo ancora il troppo lungo output, mediante l'opzione `--query` ed estraiamo la *Description* di ogni elemento delle *Images[]*:

```
$ aws ec2 describe-images --filters "Name=description,Values=*Ubuntu*22.04*amd64*2022-04*" "Name=architecture,Values=x86_64" --query Images[].Description
```

Modifichiamo il filtro sulla description e aggiungiamo alla query l'estrazione dell'*image-id*:

```
$ aws ec2 describe-images --filters "Name=description,Values=*Ubuntu*, 22.04*amd64*2022-04*" "Name=architecture,Values=x86_64" --query Images[][Description,ImageId]
```

Ancora troppi risultati! Nella query, accettiamo un indice *j* dell'array *Images[]* solo se *Images[j].Description* **non** contiene *daily* (NB: la stringa *daily* va racchiusa tra accenti nel parametro JMES di `--query`):

```
$ aws ec2 describe-images --filters "Name=description,Values=*Ubuntu*, 22.04*amd64*2022-04*" "Name=architecture,Values=x86_64" --query 'Images[?!contains(Description, `daily`)].[Description,ImageId]'
```

```
[
  [
    "Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2022-04-20",
    "ami-09d56f8956ab235b3"
  ]
]
```

L'opzione --filters

È stata già usata nei comandi precedenti. Modifichiamo il filtro sulla description e aggiungiamo alla query l'estrazione dell'*image-id*:

```
$ aws ec2 describe-images --filters "Name=description,Values=*Ubuntu\, 22.04*amd64*2022-04*"
"Name=architecture,Values=x86_64" --query 'Images[?!contains(Description, `daily`)].
[Description,ImageId] '
[
  [
    "Canonical, Ubuntu, 22.04 LTS, amd64 jammy image build on 2022-04-20",
    "ami-09d56f8956ab235b3"
  ]
]
```

L'opzione --query

Per maggiori dettagli sui filtri lato server (--filters) e lato client (--query), v.

<https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-filter.html> e

<https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-filter.html#cli-usage-filter-client-side>.

In generale, le query sono formulate in JMES, un query language per JSON (<https://jmespath.org>). Le query JMES possono facilmente diventare complesse, cf., p.es, <https://docs.rackspace.com/blog/how-to-use-the-aws-cli-to-find-untagged-instances>.

Un ausilio per tale complessità può essere fornito dal *cli-auto-prompt*

(`export AWS_CLI_AUTO_PROMPT=on` o `aws --cli-auto-prompt ...`). Un altro efficace ausilio (dopo un eventuale `unset AWS_CLI_AUTO_PROMPT`) è fornito dal tool *jpterm*:

```
$ aws ec2 describe-images -output json | jpterm
```

che mostrerà l'output JSON di `aws ec2 ...` in una schermata interattiva in cui si possono applicare e sperimentare query JMES. **NB**: l'eseguibile *jpterm* è parte del package Python *jmespath-terminal* da installare con:

```
$ pip install --user jmespath-terminal
```

(vedere <https://github.com/jmespath/jmespath-terminal> e <https://www.geeksforgeeks.org/how-to-install-jmespath-terminal-on-linux/>).

Altri tool per visualizzare efficacemente l'output json sono Visual Studio Code, *jp*

(<https://github.com/cburgmer/jp>) e *jq*:

```
$ aws ec2 describe-images -output json | jq -C
```

Usare la AWS CLI

Vediamo alcuni elementi e pattern comuni nell'uso della AWS Command Line Interface.

NB: la AWS CLI esegue chiamate API ai servizi AWS via HTTPS, quindi dal cliente devono essere abilitate le connessioni in uscita verso il port TCP 443.

Ottenere aiuto con la AWS CLI

Come già visto, autocompletamento e autoprompt sono strumenti potenti ed efficaci per guidare l'operatore, anche meno esperto, ma vanno installati (cosa non sempre semplice o possibile).

Una forma di aiuto sempre disponibile, dato un comando AWS CLI, anche parziale, si ottiene aggiungendo `help` alla fine del comando stesso, come si vede da:

```
$ aws
usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:
  aws help
  aws <command> help
  aws <command> <subcommand> help
```

P.es., per elencare le opzioni generiche e i comandi AWS top-level disponibili:

```
$ aws help
```

Si ottiene una sorta di pagina *man*. Fornendo (volutamente) un comando inesistente:

```
$ aws noncomando
usage: aws [options] <command> <subcommand> [<subcommand> ...] [parameters]
To see help text, you can run:
  aws help
  aws <command> help
  aws <command> <subcommand> help
aws: error: argument command: Invalid choice, valid choices are:
acm                               | apigateway
...                               | ...
dynamodb                         | dynamodbstreams
ec2                               | ecr
ecs                               | efs
...                               | ...
```

Elencare i sottocomandi del servizio `ec2`:

```
$ aws ec2 help
...
```

Tra i sottocomandi elencati figura `describe-instances`:

```
$ aws ec2 describe-instances help
```

L'informazione che si otterrà (per questo ed altri comandi) è una sorta di *man* page, strutturata in sei sezioni:

Name – del comando

NAME

describe-instances -

Description – dell'operazione API corrispondente al comando (proviene dalla documentazione della API)

DESCRIPTION

Describes one or more of your instances.

If you specify one or more instance IDs, Amazon EC2 returns information for those instances. If you do not specify instance IDs, Amazon EC2 returns information for all relevant instances. If you specify an instance ID that is not valid, an error is returned. If you specify an instance that you do not own, it is not included in the returned results.

...

Synopsis – elenca le opzioni del comando

SYNOPSIS

```
describe-instances
[--dry-run | --no-dry-run]
[--instance-ids <value>]
[--filters <value>]
[--cli-input-json <value>]
[--starting-token <value>]
[--page-size <value>]
[--max-items <value>]
[--generate-cli-skeleton]
```

Options – descrizione del comportamento richiesto con ciascuna delle opzioni elencate nella sezione *Synopsis*

OPTIONS

```
--dry-run | --no-dry-run (boolean)
    Checks whether you have the required permissions for the action,
    without actually making the request, and provides an error response.
    If you have the required permissions, the error response is DryRun-
    Operation . Otherwise, it is UnauthorizedOperation .

--instance-ids (list)
    One or more instance IDs.

    Default: Describes all your instances.
```

...

Per lo più, l'opzione è seguita da un parametro, che ha un tipo. La specifica dei parametri è discussa più avanti.

Examples – mostrano l'uso tipico del comando con le sue opzioni

EXAMPLES

To describe an Amazon EC2 instance

Command:

```
aws ec2 describe-instances --instance-ids i-5203422c
```

To describe all instances with the instance type m1.small

Command:

```
aws ec2 describe-instances --filters "Name=instance-type,Values=m1.small"
```

To describe all instances with a *Owner* tag

Command:

```
aws ec2 describe-instances --filters "Name=tag-key,Values=Owner"
```

...

In **EXAMPLES** viene in genere visualizzato anche un “sample output”, ovviamente conforme alla struttura descritta nella successiva sezione **OUTPUT** (v. sotto).

Output – descrizione della struttura dell’output ottenuto in risposta al comando AWS

P.es. per `describe-instances`, l’output è una lista di *reservations*, ognuna delle quali descrive una “*launch request*”, la richiesta di lancio in esecuzione di un’istanza o (come è possibile) gruppo di istanze. Lo si ricava dalla sezione **OUTPUT**:

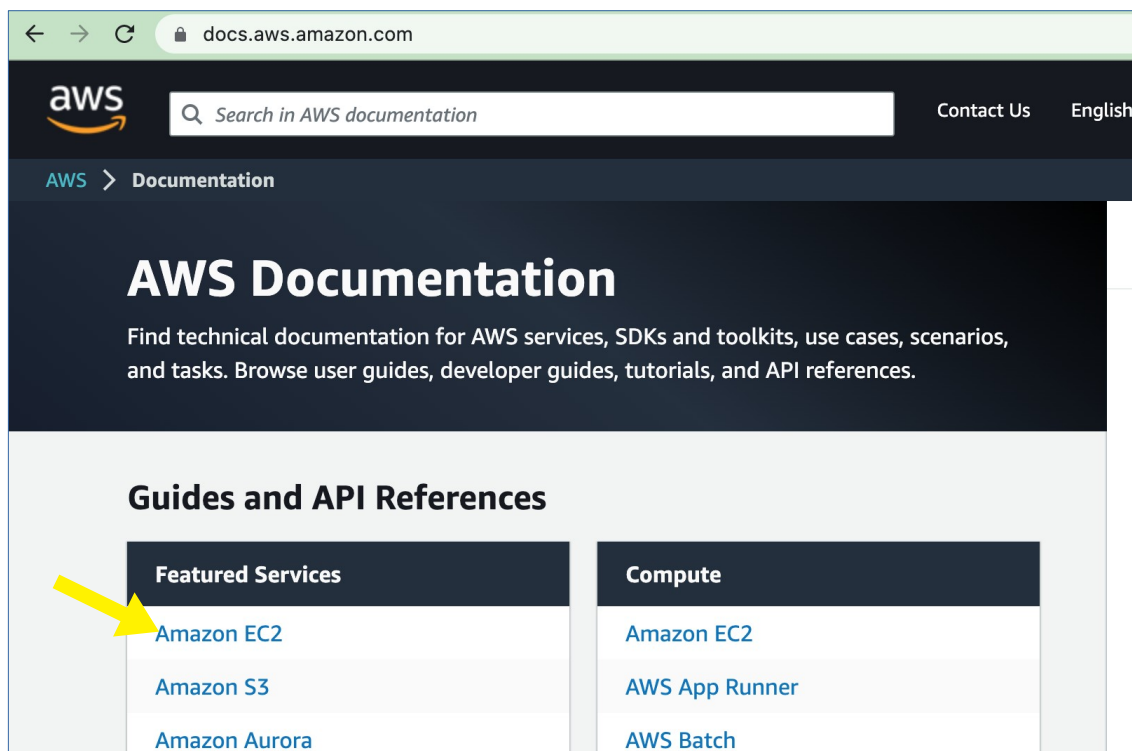
```
OUTPUT
  Reservations -> (list)
    Information about the reservations.

    (structure)
      Describes a launch request for one or more instances
  ...
```

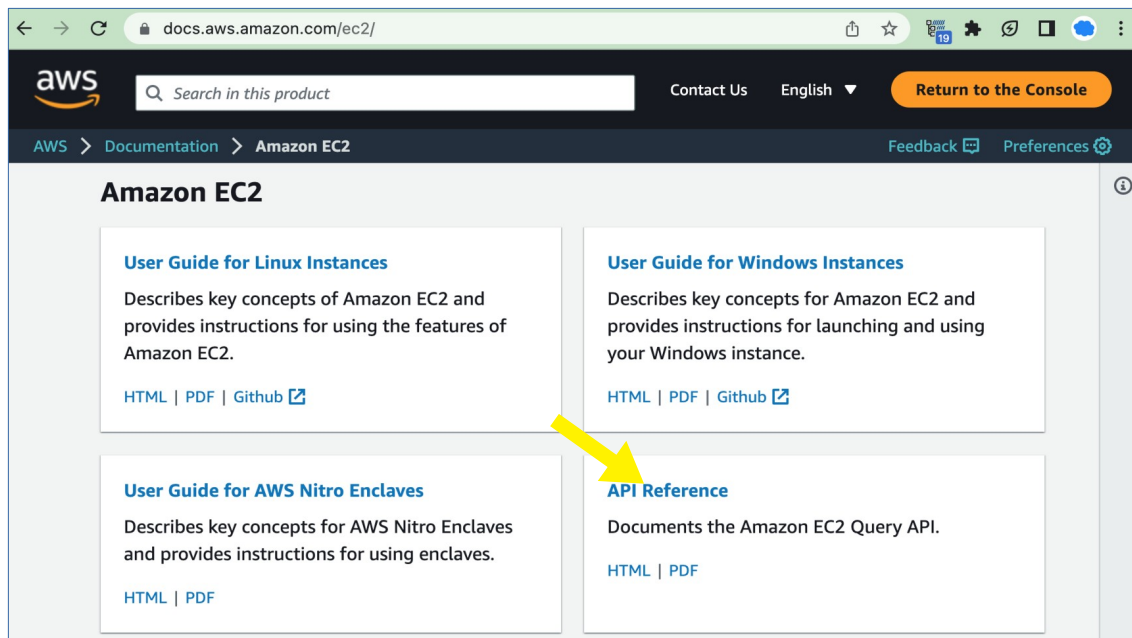
Si noti come **reservation** venga usato come una sorta di (nome di) tipo, sebbene non esplicitamente introdotto, del quale viene infatti poi descritta la *structure*. In effetti, le convenzioni usate per descrivere il formato dei dati **OUTPUT** non sono né chiarissime, né eleganti. Faremo affidamento sull’intuizione e sulla documentazione della API AWS, in quanto a ogni tipo di dato introdotto informalmente nella sezione **OUTPUT** corrisponde in effetti un tipo di dato “ufficiale” della API.

I Data Types della API AWS

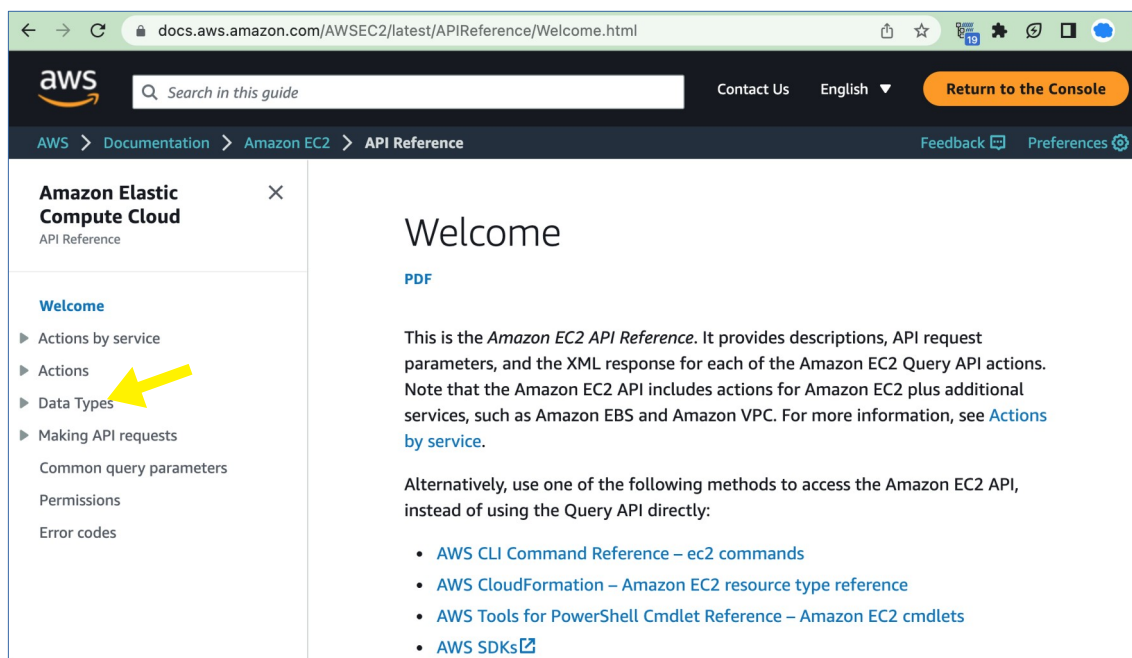
Infatti, la documentazione API AWS/EC2 prevede numerosi tipi di dato, tra cui *Reservation*. Scopriamoli partendo da <https://docs.aws.amazon.com>:



quindi <https://docs.aws.amazon.com/ec2/>:



quindi: <https://docs.aws.amazon.com/AWSEC2/latest/APIReference/>:



e, dall'elenco *Data Types*, si troverà *Reservation*:

https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_Reservation.html

Output di un comando CLI

Dopo la digressione sui tipi di dato della API AWS, torniamo alla CLI.

Come si vede, la notazione impiegata per descrivere la struttura dell'output vorrebbe essere "astratta", cioè indipendente dal formato che potrà assumere l'output (che sia JSON, testo, etc.). Per comprenderla, si tenga presente una sorta di corrispondenza tra convenzioni/notazione della pagina di help e JSON:

- Una scrittura come `Reservations -> ...` indica una coppia *chiave/valore* (o *nome/valore*), cioè `"Reservations" / ...`; in JSON la si denoterebbe con: `"Reservations": ...`
- Le parentesi tonde `()` racchiudono una sorta di tipo
- `(structure)` corrisponde a un *Object* del JSON; è, cioè, una collezione di coppie *chiave/valore*
- `(list)` corrisponde a un Array del JSON
- `(integer)`, `(string)`, `(boolean)` hanno l'ovvio significato

Come esempio, si confronti la seguente sezione OUTPUT dell'*help* di `describe-instances` con il riquadro ancora successivo, tratto sempre dallo stesso *help*.

OUTPUT

```
Reservations -> (list)
  One or more reservations.

  (structure)
    Describes a reservation.

    ReservationId -> (string)
      The ID of the reservation.

    OwnerId -> (string)
      The ID of the AWS account that owns the reservation.

    RequesterId -> (string)
      The ID of the requester that launched the instances on your
      behalf (for example, AWS Management Console or Auto Scaling).

    Groups -> (list)
      One or more security groups.

      (structure)
        Describes a security group.

        GroupName -> (string)
          The name of the security group.

        GroupId -> (string)
          The ID of the security group.

    Instances -> (list)
      One or more instances.

      (structure)
        Describes an instance.

        InstanceId -> (string)
          The ID of the instance.

        ImageId -> (string)
          The ID of the AMI used to launch the instance.

        State -> (structure)
          The current state of the instance.

          Code -> (integer)
            The state of the instance as a 16-bit unsigned
...

```

Ecco, tratto sempre dall'*help* in linea per `describe-instances`, un esempio di output conforme alla sintassi descritta sopra:

```
{
  "Reservations": [
    {

```

```

"ReservationId": "r-1234567890abcdefg",
"OwnerId": "111111111111",
"RequesterId": "111111111111",
"Groups": [],
"Instances": [
  {
    "InstanceId": "i-1234567890abcdef0",
    "ImageId": "ami-0abcdef1234567890",
    "State": {
      "Code": 16,
      "Name": "running"
    },
    "InstanceType": "t3.nano",
    "KeyName": "my-key-pair",
    "LaunchTime": "2022-11-15T10:48:59+00:00",
    "Monitoring": {
      "State": "disabled"
    },
    "Placement": {
      "AvailabilityZone": "us-east-2a",
      "GroupName": "",
      "Tenancy": "default"
    },
    "PrivateDnsName": "ip-10-0-0-157.us-east-2.compute.internal",
    "PrivateIpAddress": "10-0-0-157",
    "PublicDnsName": "ec2-34-253-223-13.us-east-2.compute.amazonaws.com",
    "PublicIpAddress": "34.253.223.13",
    "SubnetId": "subnet-04a636d18e83cfacb",
    "VpcId": "vpc-1234567890abcdef0",
    "Architecture": "x86_64",
    ...
  }
]
}

```

Documentazione AWS CLI online

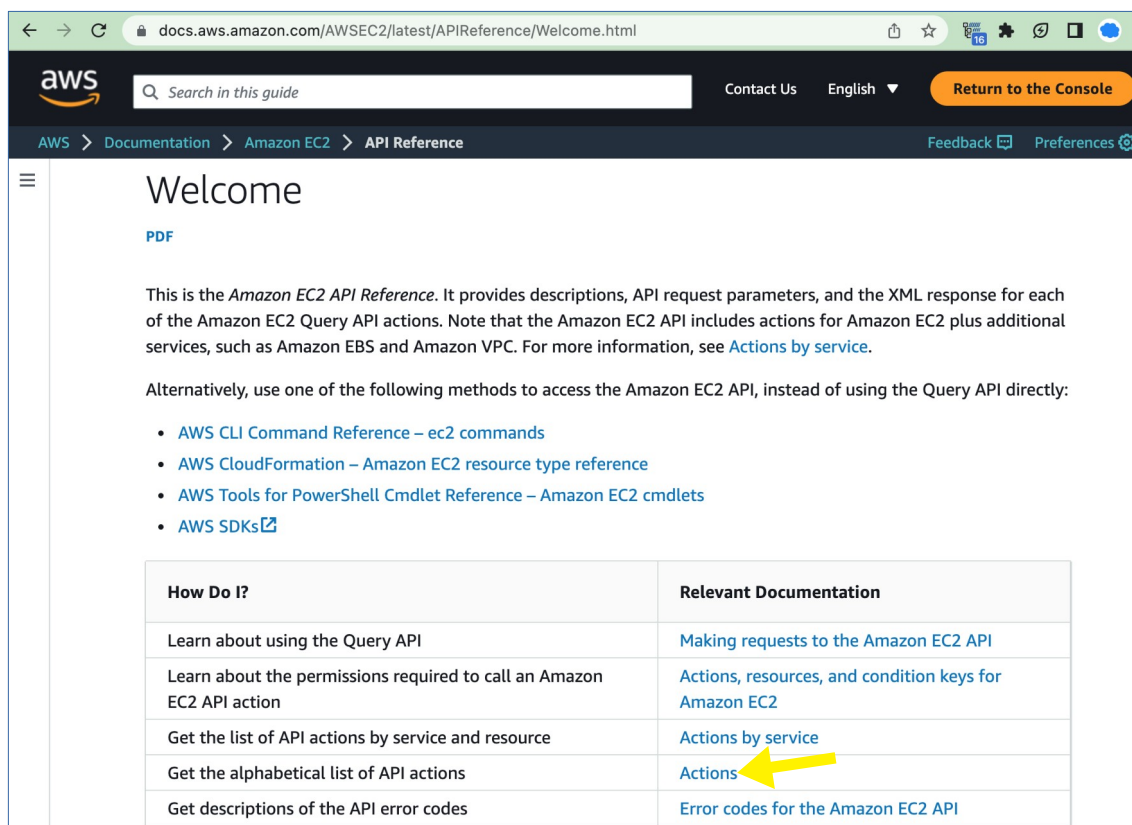
Alla URL <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/index.html> si trova la **AWS Command Line Interface Reference** (versione 2), che fornisce lo stesso contenuto degli help da riga di comando, presentato online attraverso il browser, con un bonus: alcuni link inattivi nell'help a riga di comando sono invece funzionanti nella Reference online.

Si veda, come esempio, <https://awscli.amazonaws.com/v2/documentation/api/latest/reference/ec2/describe-instances.html>

Documentazione API

Per ciascuna chiamata della API pubblica di un qualche servizio AWS, esiste un corrispondente comando/sottocomando/etc. della CLI AWS. La documentazione di riferimento di ciascuna API può essere trovata, come mostrato sopra, a partire dalla URL <https://docs.aws.amazon.com>. Per EC2 si perviene così a:

<https://docs.aws.amazon.com/AWSEC2/latest/APIReference/Welcome.html> (si veda **How Do I?**) e si clicchi su [Actions](#) (abbiamo già discusso la sezione [Data Types](#))



The screenshot shows the AWS EC2 API Reference Welcome page. The page has a dark header with the AWS logo, a search bar, and navigation links. The main content area is white and contains a 'Welcome' section with a PDF icon. Below this is a paragraph explaining the Amazon EC2 API Reference and a list of links to other documentation. At the bottom, there is a table titled 'How Do I?' with two columns: 'How Do I?' and 'Relevant Documentation'. A yellow arrow points to the 'Actions' link in the 'Relevant Documentation' column.

How Do I?	Relevant Documentation
Learn about using the Query API	Making requests to the Amazon EC2 API
Learn about the permissions required to call an Amazon EC2 API action	Actions, resources, and condition keys for Amazon EC2
Get the list of API actions by service and resource	Actions by service
Get the alphabetical list of API actions	Actions
Get descriptions of the API error codes	Error codes for the Amazon EC2 API

Infine, scegliendo l'azione di API *DescribeInstances*

(https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_DescribeInstances.html):

PDF

If you describe instances in the rare case where an Availability Zone is experiencing a service disruption and you specify instance IDs that are in the affected zone, or do not specify any instance IDs at all, the call fails. If you describe instances and specify only instance IDs that are in an unaffected zone, the call works normally.

The following parameters are for this specific action. For more information about required and



Struttura dei comandi di AWS CLI

```
$ aws <command> <subcommand> [options and parameters]
```

Come già visto, la struttura dei comandi AWS CLI consta di più parti:

- chiamata base al programma `aws`
- comando *top-level*, che per lo più rappresenta un servizio AWS
- ogni comando/(servizio AWS) ha sottocomandi aggiuntivi che specificano l'operazione da compiere
- seguono, in ordine arbitrario, opzioni generali della CLI e parametri specifici dell'operazione;
se uno di questi viene inserito più volte in un comando, ma ne è prevista una sola occorrenza, AWS CLI considera solo l'ultima.
- i parametri "di input" possono assumere valori di diversi tipi, quali numeri, stringhe, liste, mappe e strutture JSON.

Specificare i parametri di AWS CLI

Molti parametri sono semplici stringhe o valori numerici, come l'identificatore *my-key-pair* in questo esempio:

```
$ aws ec2 create-key-pair --key-name my-key-pair
```

Se un parametro stringa contiene spazi, deve essere quotato (il che è possibile anche in assenza di spazi):

Windows PowerShell, Linux, macOS, or Unix

```
$ aws ec2 create-key-pair --key-name 'my key pair'
```

Windows Command Interpreter

```
> aws ec2 create-key-pair --key-name "my key pair"
```

Il valore di input può essere collegato all'opzione anche da un segno di uguale, piuttosto che da uno spazio. Quest'accorgimento è indispensabile se il valore inizia con un segno meno (nell'esempio il nome è *"-mykey"* (segno *"-"* compreso)):

```
$ aws ec2 delete-key-pair --key-name=-mykey
```

Tipi di parametri tipici e loro specifica

A parte le generalità qui indicate, si può ottenere aiuto specifico attraverso l'*help*. Per esempio, per conoscere parametri e tipi ammessi per un sottocomando:

```
$ aws ec2 describe-spot-price-history help
```

Nell'aiuto così ottenuto, la sezione **Options** elencherà opzioni/parametri e il loro tipo.

Vediamo allora i casi che possono presentarsi per il tipo di un parametro.

String – String parameters can contain alphanumeric characters, symbols, and whitespace from the ASCII character set. Strings that contain whitespace must be surrounded by quotes. Use of symbols and whitespace other than the standard space character is not recommended and may cause issues when using the AWS CLI.

Timestamp – Secondo lo standard ISO 8601 standard. P.es.:

```
$ aws ec2 describe-spot-price-history --start-time 2014-10-13T19:00:00Z
```

Formati accettabili:

- YYYY-MM-DDThh:mm:ss.sssZ (UTC), e.g., 2014-10-01T20:30:00.000Z
- YYYY-MM-DDThh:mm:ss.sss±[hh]:[mm] (with offset), e.g., 2014-10-01T12:30:00.000-08:00
- YYYY-MM-DD, e.g., 2014-10-01

- Unix time in seconds, e.g. 1412195400

List – Una o più stringhe separate da spazi.

```
$ aws ec2 describe-spot-price-history --instance-types m1.xlarge m1.medium
```

Boolean – Flag binario che “accende” o “spegne” un’opzione. Per esempio:

```
$ aws ec2 describe-spot-price-history --dry-run
```

Qui l’opzione *--dry-run* fa sì che la query non venga in realtà effettuata. Viene solo verificato che la riga di comando sia formata correttamente. Esiste anche l’opzione *--no-dry-run*, anche se dà luogo a comportamento di default.

Integer – Il parametro dell’opzione è un numero intero senza segno. P.es.:

```
$ aws ec2 describe-spot-price-history --max-items 5
```

Blob – Il parametro dell’opzione è un oggetto binario. Si tratta del contenuto di un file “locale” (del cliente su cui gira AWS CLI). Il parametro *blob* è quindi specificato dal pathname del file che contiene i dati, senza protocol identifier come *http://* o *file://*. P. es.:

```
$ aws s3api put-object --bucket my-bucket --key testimage.png --body /tmp/image.png
```

Map – Il parametro dell’opzione è una sequenza di coppie chiave-valore, specificate in JSON (trattato oltre) o in *sintassi abbreviata*.

Nell’esempio seguente, il parametro *--key* è una *map* che ha chiave *id*, valore che è una mappa a sua volta, questa con valore numerico 1. Come si vede il parametro e l’output sono specificati in JSON:

```
$ aws dynamodb get-item --table-name my-table --key '{"id": {"N": "1"}}'
```

```
{
  "Item": {
    "name": {
      "S": "John"
    },
    "id": {
      "N": "1"
    }
  }
}
```

Sintassi abbreviata al posto di JSON

Consideriamo un esempio di opzione in cui il valore, specificato in JSON, sia un elenco a un solo livello:

```
--some-option '{"key1":"value1", "key2":"value2", "key3":"value3"}'
```

In questo caso, la sintassi JSON (graffe, due punti, virgolette...) può essere una complicazione inutile. È possibile utilizzare una forma abbreviata, non-JSON, per una lista di coppie nome-valore, come specificato di seguito.

Linux, macOS, or Unix

```
--some-option key1=value1,key2=value2,key3=value3
```

Windows PowerShell

```
--option "key1=value1,key2=value2,key3=value3"
```

NB: nessuno spazio dopo la virgola che separa le varie coppie.

Un altro esempio:

```
$ aws dynamodb update-table --provisioned-throughput ReadCapacityUnits=15,WriteCapacityUnits=10  
--table-name MyDDBTable
```

Lo stesso, in JSON:

```
$ aws dynamodb update-table --provisioned-throughput  
'{"ReadCapacityUnits":15,"WriteCapacityUnits":10}' --table-name MyDDBTable
```

Parametri lista, abbreviati e in JSON

In sintassi abbreviata, il valore lista dell'opzione `--some-option` si specifica separando gli elementi con spazi:

```
--some-option value1 value2 value3
```

In JSON:

```
--some-option '[value1,value2,value3]'
```

Esempio con lista di stringhe, notazione semplificata:

```
$ aws ec2 stop-instances --instance-ids i-1486157a i-1286157c i-ec3a7e87
```

La stessa in JSON:

```
$ aws ec2 stop-instances --instance-ids '["i-1486157a","i-1286157c","i-ec3a7e87"]'
```

Struttura complessa, in notazione abbreviata (lista di coppie nome/valore):

```
$ aws ec2 create-tags --resources i-1286157c --tags Key=My1stTag,keyA=ValueA  
Key=My2ndTag,keyB=ValueB Key=My3rdTag,keyC=ValueC
```

In JSON (su più righe per leggibilità):

```
$ aws ec2 create-tags --resources i-1286157c --tags '[  
  {"Key": "My1stTag", "keyA": "ValueA"},  
  {"Key": "My2ndTag", "keyB": "ValueB"},  
  {"Key": "My3rdTag", "keyC": "ValueC"}  
]
```

Gestire i parametri JSON

Esempio - comando EC2 che elenca le istanze con dei filtri: quelle che hanno instance type m1.small o m1.medium e che si trovano nella Availability Zone us-west-2c:

```
$ aws ec2 describe-instances --filters Name=instance-type,Values=t2.micro,m1.medium  
Name=availability-zone,Values=us-west-2c
```

Vediamo lo stesso parametro valore dell'opzione `--filters` in JSON. Si noti che il parametro è un array (racchiuso nelle quadre più esterne), ogni elemento è una coppia (racchiusa in graffe) e il secondo elemento della coppia è una map che ha per valore un array (ancora tra quadre).

```
[  
  {  
    "Name": "instance-type",  
    "Values": ["t2.micro", "m1.medium"]  
  },  
  {  
    "Name": "availability-zone",  
    "Values": ["us-east-1c"]  
  }  
]
```

N.B.: il valore della chiave "Values" dev'essere un array (tra quadre) anche se questo contenesse un solo elemento. Tuttavia le quadre più esterne sono richieste solo se il parametro contiene più coppie; se ve n'è una sola (cioè un solo filtro), come qui:

```
[  
  {  
    "Name": "instance-type",  
    "Values": ["t2.micro", "m1.medium"]  
  },  
]
```

basterano le graffe:

```
$ aws ec2 describe-instances --filters '{"Name": "instance-type", "Values": ["t2.micro",  
"m1.medium"]}'
```

Si noti come, sulla riga di comando, occorre quotare il parametro JSON.

Un altro esempio:

```
$ aws ec2 run-instances --block-device-mappings <list-of-block-mappings>
```

Un *block mapping* da blocco EBS di 20GB a device logica `/dev/sdb` di un istanza EC2, specificato in JSON come lista di due coppie chiave/valore:

```
{
  "DeviceName": "/dev/sdb",
  "Ebs": {
    "VolumeSize": 20,
    "DeleteOnTermination": false,
    "VolumeType": "standard"
  }
}
```

Per avere più device, i rispettivi *block mapping* vengono elencati in una lista, che in JSON si rende come array (racchiuso tra quadre)

```
[
  {
    "DeviceName": "/dev/sdb",
    "Ebs": {
      "VolumeSize": 20,
      "DeleteOnTermination": false,
      "VolumeType": "standard"
    }
  },
  {
    "DeviceName": "/dev/sdc",
    "Ebs": {
      "VolumeSize": 10,
      "DeleteOnTermination": true,
      "VolumeType": "standard"
    }
  }
]
```

Il parametro JSON si può passare sulla riga di comando, quotandolo opportunamente, come spiegato di seguito, oppure lo si può leggere da un file. Per definizioni lunghe, il file è più semplice da gestire (leggere, condividere, modificare)

Quoting Strings

The way you enter JSON-formatted parameters on the command line differs depending upon your operating system. Linux, macOS, or Unix and Windows PowerShell use the single quote (') to enclose the JSON data structure, as in the following example:

```
$ aws ec2 run-instances --image-id ami-05355a6c --block-device-mappings
'[{ "DeviceName": "/dev/sdb", "Ebs": {
  "VolumeSize": 20, "DeleteOnTermination": false, "VolumeType": "standard" } } ]'
```

The Windows command prompt, on the other hand, uses the double quote (") to enclose the JSON data structure. In addition, a backslash (\) escape character is required for each double quote (") within the JSON data structure itself, as in the following example:

```
C:\> aws ec2 run-instances --image-id ami-05355a6c --block-device-mappings "[{ \"DeviceName\":
\"/dev/sdb\", \"Ebs\": { \"VolumeSize\": 20, \"DeleteOnTermination\": false, \"VolumeType\":
\"standard\" } } ]"
```

Windows PowerShell requires a single quote (') to enclose the JSON data structure, as well as a backslash (\) to escape each double quote (") within the JSON structure, as in the following example:

```
> aws ec2 run-instances --image-id ami-05355a6c --block-device-mappings
'[{\"DeviceName\": \"\\dev/sdb\\\", \"Ebs\": {\"VolumeSize\": 20, \"DeleteOnTermination\": false,
\\\"VolumeType\\\": \"standard\\\"}}]'
```

If the value of a parameter is itself a JSON “document”, escape the quotes on the embedded JSON document. For example, the attribute parameter for `aws sqs create-queue` can take a `RedrivePolicy` key. The value of `RedrivePolicy` is a JSON document, which must be escaped:

```
$ aws sqs create-queue --queue-name my-queue --attributes
'{ \"RedrivePolicy\": \"{\\\"deadLetterTargetArn\\\": \"arn:aws:sqs:us-west-2:0123456789012:deadletter\\\",
\\\"maxReceiveCount\\\": 5\\\"}\"}'
```

Loading Parameters from a File

To avoid the need to escape JSON strings at the command line, load the JSON from a file. Load parameters from a local file by providing the path to the file using the `file://` prefix, as in the following examples.

Linux, macOS, or Unix

```
// Read from a file in the current directory
$ aws ec2 describe-instances --filters file://filter.json

// Read from a file in /tmp
$ aws ec2 describe-instances --filters file:///tmp/filter.json
```

Windows

```
// Read from a file in C:\temp
> aws ec2 describe-instances --filters file://C:\temp\filter.json
```

The `file://` prefix option supports Unix-style expansions including `'~/`, `'./`, and `'../`. On Windows, the `'~/` expression expands to your user directory, stored in the `%USERPROFILE%` environment variable. For example, on Windows 7 you would typically have a user directory under `C:\Users\User Name\`.

JSON documents that are provided as the value of a parameter key must still be escaped:

```
$ aws sqs create-queue --queue-name my-queue --attributes file://attributes.json
```

attributes.json

```
{
  "RedrivePolicy": "{ \"deadLetterTargetArn\": \"arn:aws:sqs:us-west-2:0123456789012:deadletter\",
  \"maxReceiveCount\": \"5\" }",
}
```


Binary Files

For commands that take binary data as a parameter, specify that the data is binary content by using the `fileb://` prefix. Commands that accept binary data include:

- **aws ec2 run-instances** takes the `-user-data` parameter.
- **aws s3api put-object** takes the `--sse-customer-key` parameter.
- **aws kms decrypt** takes the `-ciphertext-blob` parameter.

The following example generates a binary 256 bit AES key using a Linux command line tool and then provides it to Amazon S3 to encrypt an uploaded file server-side:

```
$ dd if=/dev/urandom bs=1 count=32 > sse.key
32+0 records in
32+0 records out
32 bytes (32 B) copied, 0.000164441 s, 195 kB/s
$ aws s3api put-object --bucket my-bucket --key test.txt --body test.txt --sse-customer-key
fileb://sse.key --sse-customer-algorithm AES256
{
  "SSECustomerKeyMD5": "iVg8oWa8sy714+FjtesrJg==",
  "SSECustomerAlgorithm": "AES256",
  "ETag": "\"a6118e84b76cf98bf04bbe14b6045c6c\""
}
```

Remote Files

The AWS CLI also supports loading parameters from a file hosted on the Internet with an `http://` or `https://` URL. The following example references a file in an Amazon S3 bucket (i bucket sono accessibili tramite URL).

This allows you to access parameter files from any computer, but requires the file to be stored in a publically accessible location.

```
$ aws ec2 run-instances --image-id ami-a13d6891 --block-device-mappings http://my-bucket.s3.amazonaws.com/filename.json
```

In the preceding examples, the `filename.json` file contains the following JSON data.

```
[
  {
    "DeviceName": "/dev/sdb",
    "Ebs": {
      "VolumeSize": 20,
      "DeleteOnTermination": false,
      "VolumeType": "standard"
    }
  }
]
```

For another example referencing a file containing more complex JSON-formatted parameters, see [Set an IAM Policy for an IAM User](#).