Hands-on Cloud Computing Services

Lezione 3

Gabriele Russo Russo

University of Rome Tor Vergata, Italy

A.A. 2024/25

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Recap

- We have seen how to deploy a web app using:
  - EC2
  - ELB
  - Auto Scaling Groups
- Problem: infrastructure completely configured by hand
  - error-prone and difficult to reproduce

AWS CLI

- Command Line Interface to interact with AWS
- Faster interaction compared to web console
  - e.g., EC2 instance created with a single command
- Installation: check the official docs for Linux/Win/macOS
  - `https://aws.amazon.com/it/cli/`
- AWS CloudShell provides an in-browser console where CLI commands are available (useful for quick commands)
- Alternatively, Windows users may prefer the AWS Tools for PowerShell
  - `https://aws.amazon.com/it/powershell/`

AWS CLI: Configuration

- AWS CLI can be configured by:
  - running `aws configure`, or
  - editing `~/.aws/config` and `~/.aws/credentials`
  - (slightly different paths on Windows)
- Key configuration options:
  - `AWS Access Key ID` and `AWS Secret Access Key`
  - default region to use (e.g., `us-east-1`)
  - output format (`json`, `text`)

AWS CLI: example (1)

Create a new security group in our VPC:

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

Set inbound traffic rules, e.g.:

```
$ aws ec2 authorize-security-group-ingress \
         --group-id <ID> \
         --protocol tcp --port 22 --cidr 0.0.0.0/0
```

We can see the properties of any SG:

```
$ aws ec2 describe-security-groups --group-ids <groupId>
```

AWS CLI: example (2)

Create an EC2 instance:

```
$ aws ec2 run-instances \
        --image-id <ID AMI> \
        --count 1 \
        --instance-type t2.nano \
        --key-name <MyKeyPair> \
        --security-group-ids <sgId> \
        --subnet-id <subnetId> \
        --associate-public-ip-address
```

We can associate the instance with a tag:

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

AWS CLI: example (3)

We can get information about active instances:

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

To terminate the instance:

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

Exercises

- Create a script to destroy all the active EC2 instances
- Create a script to destroy all the active EC2 instances with tag “Name=SDCC”

Recap

- AWS CLI (partially) solves the problem of automating cloud infrastructure management
  - scripts better than manual web-based management
  - (not the best solution though)
- How to automate application deployment as well?

IT Automation using Ansible

- *Ansible delivers simple IT **automation** that ends repetitive tasks and frees up teams for more strategic work.*
- Available on Linux and macOS: `https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html`
  - Windows users might use a Linux-based VM
- Agentless: no need to pre-install software on the target machines
- Define WHAT you want to achieve, instead of HOW
  - e.g., “Apache web server installed and started”
- Similar alternatives: Chef, Puppet, ~~a bunch of Bash scripts~~, ...

Ansible: Key Concepts

- Playbooks (e.g., “deploy Photogallery”)
- Tasks (e.g., (“install Flask”)
- Modules (used to define single sub-tasks: e.g., file, archive, apt)
  - Built-in modules
  - Custom modules
- Inventory = hosts to be managed
  - Static
  - Dynamic

A playbook for Photogallery: inventory

- Create the inventory file `hosts.ini`
  - (You may even put `localhost` in the inventory for testing)
- One line per host
- Possibly organized into groups (e.g., web, db, ...)
- We can add params for SSH authentication on the same line

Inventory file

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

Simple test using the *ping* module:

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

A playbook for Photogallery

To deploy Photogallery we need to:

- Upload application files (module: **copy**)
- Install dependencies (modules: **yum**, **pip**)
- Install systemd unit file to start server at boot (module: **copy**)
- Enable systemd service (module: **systemd**)

Check `deploy_gallery.yaml`

```
$ ansible-playbook -v -i hosts.ini deploy_gallery.yaml
  # What happens if we try again?
$ ansible-playbook -v -i hosts.ini deploy_gallery.yaml
```

Ansible: Dynamic Inventory

- Ansible requires an inventory
- Not necessarily a static file
- AWS Inventory Source: run your playbooks using (a subset of) your EC2 instances as target hosts (e.g., filtered by tag)
- Requires Ansible 2.9+
- A plugin required, easy to install:

```
$ ansible-galaxy collection install amazon.aws
```

Ansible: AWS Dynamic Inventory

- Create a YAML file (name MUST end with `aws_ec2.(yml|yaml)`
  → `galleryInventory.aws_ec2.yaml`

Test

```
ansible-inventory -i galleryInventory.aws_ec2.yaml --graph
```

Running the playbook

```
ansible-playbook -i galleryInventory.aws_ec2.yaml
--private-key=path/to/key.pem -u ec2-user deploy_gallery.yaml
```

Ansible: More Advanced Stuff

- Groups and Roles
- Templates
- Ansible Tower / AWX[1]
    - Share playbooks / delegate
    - Schedule workflows
    - Dashboards

[1] `https://github.com/ansible/awx`

Amazon S3

AWS Storage Services

AWS offers various storage services, including:

- S3: Simple Storage Service
- EBS: Elastic Block Storage
- EFS: Elastic File System

Amazon S3

- Amazon Simple Storage Service (S3)
- Scalable object storage service
- Pricing: `https://aws.amazon.com/it/s3/pricing/`
- **Buckets** and **objects**

S3 for Photogallery

- Let's create a bucket using S3 console
- Bucket name must be unique across all AWS regions and accounts
- We can choose who can access objects and buckets: `https://docs.aws.amazon.com/it_it/AmazonS3/latest/dev/example-bucket-policies.html`
- For Photogallery, we want everyone to read objects

We can reference an object like this:

`https://BUCKETNAME.s3.amazonaws.com/FILENAME`

Using S3 through the CLI

```
$ 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/prova.txt
```

Third-party clients also available: e.g., *s3cmd*

Hosting Static Web Content on S3

- Objects in a public bucket can be accessed through HTTP
- You can use S3 to host static web content
  - a static website
  - the frontend of a web application
- To enable web hosting on a bucket: `https://docs.aws.amazon.com/AmazonS3/latest/userguide/EnableWebsiteHosting.html`