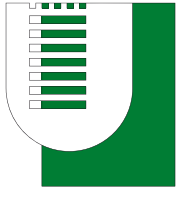
Hands-on Cloud Computing Services

Lezione 2

Gabriele Russo Russo

University of Rome Tor Vergata, Italy

A.A. 2022/23



TOR VERGATA

UNIVERSITÀ DEGLI STUDI DI ROMA

Recap

- Amazon Web Services: regions, services, ...
- Elastic Compute Cloud (EC2)
  - Instance, AMI, Security Group
  - SSH, public/private keys
- Example web app: Photogallery

Deploying Photogallery on EC2

Running Photogallery

```
$ export FLASK_APP=galleryApp.py
$ flask run -h 0.0.0.0 -p <numero di porta>
$ # Note: \-- requires root privileges for port 80
```

or, using the script `run.sh`:

```
$ bash run.sh
```

- Create a new EC2 instance to deploy the app
- Connect via SSH to the instance:

```
$ ssh -i <file.pem> ec2-user@<Public IP/Public DNS>
```

Deploying Photogallery on EC2 (contd.)

- Install the required software:

```
$ sudo yum install python3
$ sudo pip3 install flask
```

- Copy the app files from your PC using `scp`:

```
$ scp -i <chiaveprivata.pem> -r <cartellalocale> \
  ec2-user@<istanza ec2>:/home/ec2-user/
```

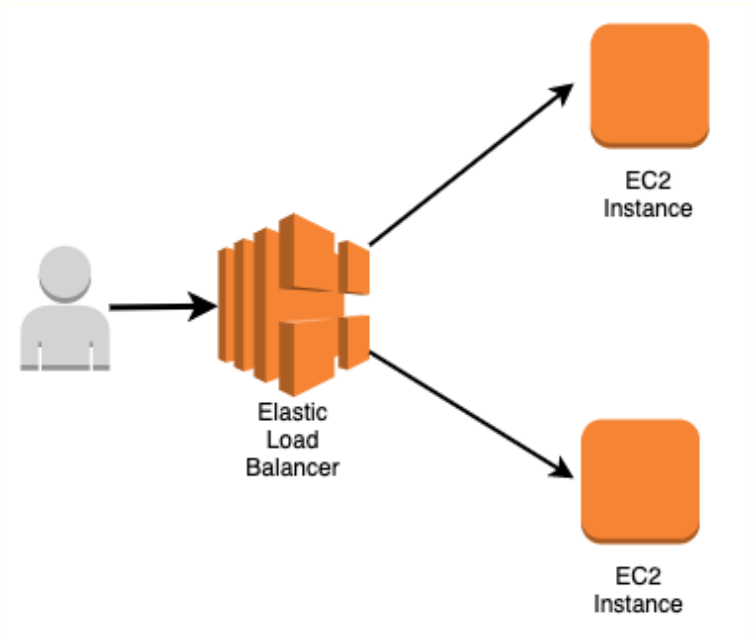
- Start the application:

```
$ cd photogallery/
$ bash run.sh
```

- Open `http://EC2-PUBLIC-IP/` in a browser
- Test: what if we “close” port 80 in the security group?

Replicating App Instances

- Current configuration is neither scalable or fault-tolerant
- Let's run multiple replicas of the web server
- We need a **load balancer**



Preliminary Tasks

- We run the app as a systemd **service**, automatically started at boot

/etc/systemd/system/photogallery.service

```
[Unit]
Description=Simple systemd service for Photogallery.

[Service]
Type=simple
WorkingDirectory=/home/ec2-user/photogallery
ExecStart=/bin/bash /home/ec2-user/photogallery/run.sh

[Install]
WantedBy=multi-user.target
```

Preliminary Tasks (contd.)

Starting and enabling the service

```
$ sudo systemctl daemon-reload
$ sudo systemctl start photogallery.service
$ sudo systemctl enable photogallery.service
```

Register an AMI

We also create an **AMI** using a snapshot of the running instance. We will be able to re-use the AMI to create new instances where the application is already installed and configured to start.

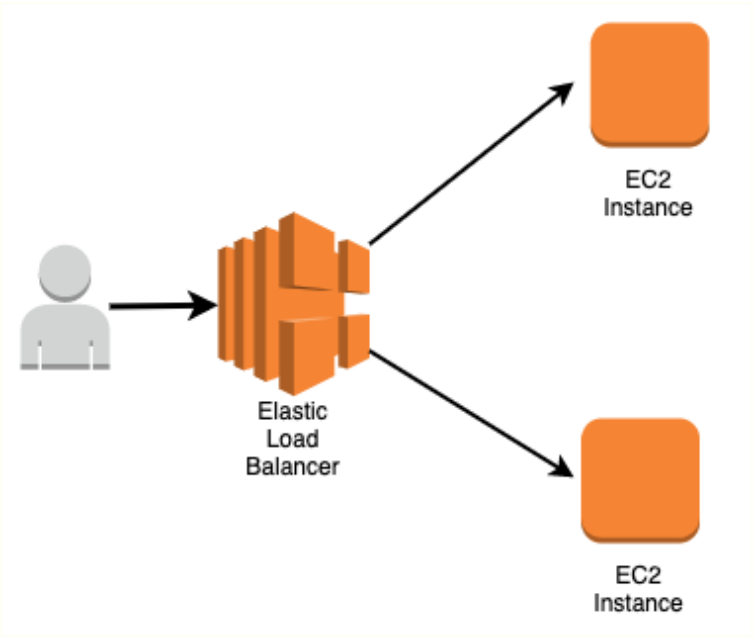
Preliminary Tasks (contd.)

Note: each AMI is associated with a snapshot of the root ELB volume attached to the instance. Keeping this snapshot has a (small) cost: `https://aws.amazon.com/premiumsupport/knowledge-center/ebs-snapshot-billing/`

Run Commands at Launch: cloud-init and User Data

- Creating a custom AMI allowed us to create new EC2 instances without manually configuring the application every time
- Any smarter approaches?
- Cloud providers allow you to run commands when instances are launched:
  `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html`
- In AWS, you can use the User Data option to specify:
  - a Bash script
  - cloud-init directives
    (`https://cloudinit.readthedocs.io/en/latest/`)

Next step



Amazon VPC

- Provision **logically isolated** sections of the AWS cloud
- Define virtual networks (IP ranges, subnets, gateways,...)
- May create a hardware Virtual Private Network (VPN) connection between your own datacenter and your VPC (**hybrid cloud**)
- **No additional charges** for creating and using the VPC itself.
- So far, we have used the default VPC

Amazon VPC: main building blocks

- In each AZ, we can define one or more **subnets**
- **Routing Tables** attached to subnets
- **Internet Gateway**

VPC Configuration: the hard way

- Create a new **Virtual Private Cloud (VPC)**
- We associate a block of (private) IP addresses to the VPC
  - Subnets will be created within this block of addressess
  - We can pick, e.g., `10.0.0.0/16`
- We can create **subnets**: each subnet is associated with an Availability Zone (AZ)
- Let's pick an AZ and create a subnet (e.g., `10.0.1.0/24`)
- If you want (for debugging), you can require that EC2 instances in the subnet are also assigned a public IP address
- Create an **Internet Gateway** (IG) to allow instances in the VPC to reach Internet; **associate** it with the VPC
- Create a **Route Table** for the VPC and **attach** it to the subnet(s)
- Add a new rule in the table: `0.0.0.0\0` – target: IG
- Repeat the above steps for each subnet you want.

VPC Configuration: the easy way

- AWS released a new UI to ease VPC configuration
- Most the elements you need automatically created along with the VPC
- You may only need to create an **Internet Gateway** (IG) to allow instances in the VPC to reach Internet and **associate** it with the VPC Add a new rule to the routing table(s): `0.0.0.0\0` – target: IG

Elastic Load Balancing (ELB)

- ELB automatically distributes incoming traffic across multiple targets (e.g., EC2 instances, containers, and IP addresses) in one or more Availability Zones
- It monitors the health of its registered targets and routes traffic only to the healthy targets
- 4 types of ELB:
  - Application Load Balancer (layer 5)
  - Network Load Balancer (layer 4)
  - Gateway Load Balancer (layer 3)
  - Classic Load Balancer (legacy)
- We'll use the Application LB today

ELB Configuration

- Create an ELB instance listening for HTTP requests on port 80
- Health check: use HTTP requests on port 80 with path /
- ELB needs a security group: configure one to accept traffic on port 80
- Create a few EC2 instances using our custom AMI in our subnets
- Register the instances to the ELB
- Wait a few minutes (DNS...) and then try to connect at the ELB URL with the browser

ELB Configuration

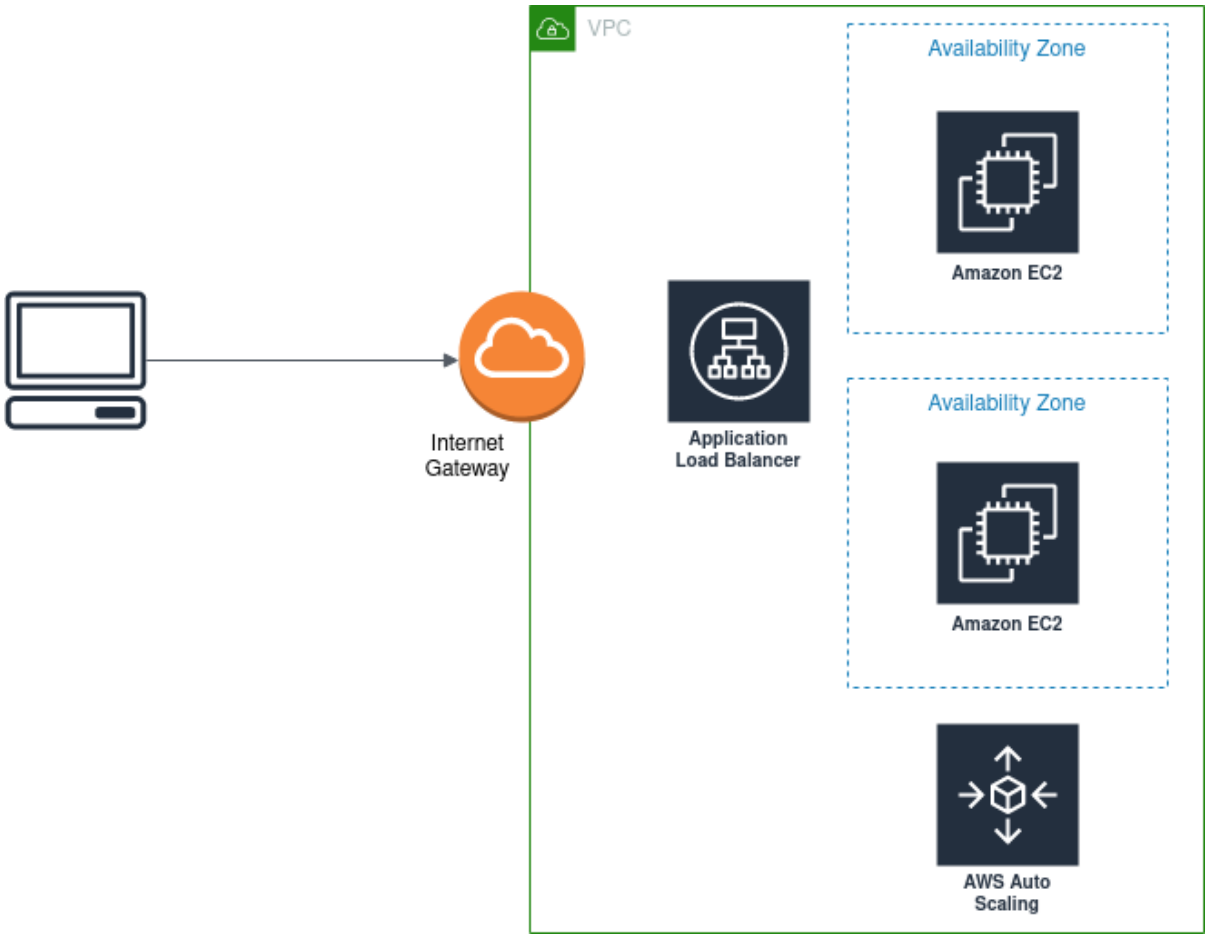
- Create an ELB instance listening for HTTP requests on port 80
- Health check: use HTTP requests on port 80 with path `/`
- ELB needs a security group: configure one to accept traffic on port 80
- Create a few EC2 instances using our custom AMI in our subnets
- Register the instances to the ELB
- Wait a few minutes (DNS...) and then try to connect at the ELB URL with the browser

Note:

- EC2 instances don't need a public IP address any more
- EC2 instances can now use a stricter security group:
  - Allowed source: `0.0.0.0/0` → `<ID of ELB sec group>`

Auto scaling

- We want to dynamically provision the number of active instances
- Let's use the Auto Scaling service of EC2



Auto Scaling + Photogallery

- Before starting, terminate manually launched instances
- Create a *Launch Template* for Photogallery
- Create an Auto Scaling Group that uses the new Launch Template
- Specify the VPC and the subnets where new instances should be launched
- Enable load balancing, associating the group with our ELB
- Set minimum and maximum number of instances (e.g., 2 and 5)
- Set an auto scaling policy
- Verify that new instances are automatically created