

Multiple Public IPs, one EC2

Below is a step-by-step guide on how to configure your AWS EC2 instance (with Proxmox installed on Debian) so that multiple Elastic IPs can be assigned to different containers or virtual machines. This assumes:

1. You already installed Proxmox on a Debian instance running in AWS.
 2. You have a bridge configured (e.g., `vmbr1` with `172.31.14.1/24`) to which you attach your containers/VMs.
 3. You know how to allocate and associate multiple Elastic IPs in the AWS console.
-

AWS Prerequisites

1. Allocate & Associate EIPs

- In the AWS console, go to **EC2** → **Elastic IPs** and allocate the addresses you need.
- Associate each Elastic IP to your EC2 instance's network interface (ENI) as a **secondary private IP**.

For example:

- EIP-1 ↔ 172.31.14.10
- EIP-2 ↔ 172.31.14.11
- EIP-3 ↔ 172.31.14.12
- EIP-4 ↔ 172.31.14.13

2. Disable Source/Dest Check

- In the **EC2** console, select your Proxmox instance → **Actions** → **Networking** → **Change source/dest. check** → set to **Disable**.
- This is crucial if you plan to forward traffic (via NAT or routing) to internal guests.

3. Security Groups

- Make sure the **Security Group** on your instance allows inbound ports you need (e.g., 22 for SSH, 80 for HTTP, 443 for HTTPS, etc.).
-

Proxmox Host Setup

1. Verify Secondary IPs on `ens33`

- Once AWS associates the private IPs (e.g., `172.31.14.10-13`), confirm they appear on the Proxmox host:

```
ip addr show ens33
```

- If they're not there, manually add them:

```
sudo ip addr add 172.31.14.10/32 dev ens33
sudo ip addr add 172.31.14.11/32 dev ens33
sudo ip addr add 172.31.14.12/32 dev ens33
sudo ip addr add 172.31.14.13/32 dev ens33
```

2. Confirm the Bridge (`vmbr1`)

- You mentioned you have `vmbr1` at `172.31.14.1/24`.
- This means the Proxmox host uses `172.31.14.1` as its IP on that bridge.
- Any container or VM assigned to `vmbr1` can then use IPs in the `172.31.14.x/24` range, with `.1` as the gateway.

Container / VM Network Configuration

1. Create a Container/VM

- In Proxmox, create or edit a container/VM and set its network device to use:
 - Bridge: `vmbr1`
 - Static IP: `172.31.14.101/24` (for example)
 - Gateway: `172.31.14.1`

2. Test Internal Connectivity

- From the host, ping `172.31.14.101`.
- Inside the container, ping `172.31.14.1`.
- Confirm that traffic flows locally on the bridge.

Making the Guest Public

AWS won't allow traditional layer-2 bridging with random MAC addresses, so we typically do **NAT** or **routed** setups:

NAT Method (Recommended)

1. Enable IP Forwarding

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Or set `net.ipv4.ip_forward=1` in `/etc/sysctl.conf`.

2. Create DNAT/SNAT Rules

- Suppose `172.31.14.10` is associated with a public EIP and you want to forward all traffic to a container at `172.31.14.101`:

```
# DNAT: traffic arriving at .10 -> container .101
iptables -t nat -A PREROUTING -d 172.31.14.10 -j DNAT --to-destination 172.31.14.101

# SNAT: traffic leaving .101 -> source it from .10
iptables -t nat -A POSTROUTING -s 172.31.14.101 -j SNAT --to-source 172.31.14.10
```

- Repeat for other secondary IPs (.11, .12, .13 → different containers).

3. Persistent iptables

- Put those rules in a script (e.g., `/root/iptables.sh`) and call it on boot (via `/etc/rc.local`, `systemd` unit, or `iptables-persistent`).

Result:

- Users connect to the public EIP, AWS maps that to `172.31.14.10`, and your Proxmox host DNATs inbound to `172.31.14.101`.
- Outbound packets from `172.31.14.101` get SNATed to `172.31.14.10`, so replies return.

Direct / Routed Approach

1. **Disable Source/Dest Check** (done).
2. **Enable `proxy_arp`** on Proxmox:

```
echo 1 > /proc/sys/net/ipv4/conf/ens33/proxy_arp
echo 1 > /proc/sys/net/ipv4/conf/vmbr1/proxy_arp
```

3. **Assign the IP inside the Guest**

- The container itself configures `172.31.14.10/24` with gateway `172.31.14.1`.

4. **Host ARP**

- The host must answer ARP for `.10` on behalf of the container (that's what `proxy_arp` does).

This method lets the container literally own the IP. However, NAT is typically easier and more common in AWS.

Final Checks & Tips

1. **Open Ports in AWS Security Group**

- Make sure inbound rules allow the ports you need for each EIP.

2. **Test Externally**

- From your local machine, try pinging or SSHing into the public EIP.
- Run `tcpdump` on the Proxmox host (`ens33`) and inside the container to confirm packet flow if debugging is needed.

3. Persist Your Config

- If you added secondary IPs manually with `ip addr add`, incorporate those changes into `/etc/network/interfaces` or a startup script.
 - If you used iptables rules, ensure they load at boot.
-

Conclusion

By disabling source/dest check, attaching multiple private IPs (each mapped to an EIP), and either using **NAT** or a **routed** approach, you can give each Proxmox container (or VM) its own unique public IP address on AWS. The **NAT** method is simplest: each container has a private IP in the `172.31.14.x/24` range, and iptables translates the traffic to/from the Proxmox host's secondary IPs. This way, you can host multiple external-facing services on a single AWS Proxmox instance.

Thanks for reading, and happy hosting!

Revision #10

Created 7 April 2025 17:44:34 by Jack Waterhouse

Updated 23 April 2025 02:59:13 by Jack Waterhouse