How to simplify your DNS management in a hybrid environment

4 February 2022 - 7 min. read

Amazon Route 53 *DNS management* *Hybrid Cloud*

DNS is a core service that keeps the internet working: the first DNS server was developed in 1984, shortly after the Internet Engineering Task Force published RFCs 882 and 883 in November 1983.

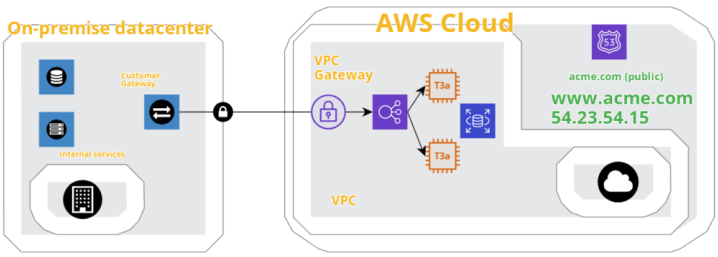
Since then, it has always done its duty in easing our life, even if its design hasn't changed a lot: new security features and record types have been added in these years to accommodate changing service needs. In the end, it's always a matter of associating a name with an IP address or another record to make it easy to remember for humans.

DNS plays a key role when using the Cloud: relying on bare IP addresses affects reliability, scalability and ease of management badly. Think about using a fleet of spot instances and having to add and remove them from a load balancer or, for worse, promoting a standby replica to a primary node in case of a failure: without a DNS record, it would be unnecessarily complicated and time-consuming.

Choosing the most appropriate way to configure and use DNS in a hybrid environment will save you time and money and help you be more resilient: sometimes internal on-premise services need to reach private cloud applications (and vice-versa). We want to accommodate this need with the tiniest operational and maintenance effort.

Let's see standard techniques and DNS configurations, as there's not a silver bullet that can solve every problem.

We'll assume that our business (ACME corp.) has an on-premises datacenter to host legacy services; the AWS Cloud is hosting newly refactored services, using a VPN connection to provide connectivity between the environments. Stay tuned for the next articles because we'll see how to centralise and consolidate networking in a complex scenario (spoiler alert: Transit Gateway and Direct Connect are coming!)
Our public DNS domain will be **acme.com**, using **Route53** to host the public zone.



We can choose to address internal services in different ways.

1. Use only the public hosted zone

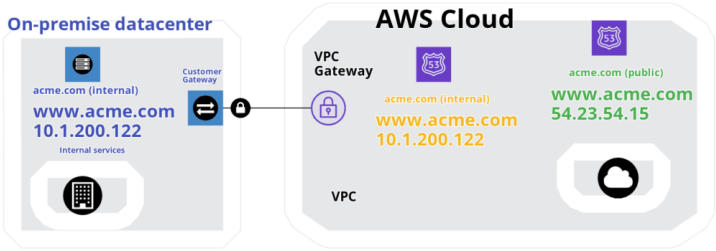
We can use the acme.com zone to resolve internal IP addresses centralising management. AWS uses this pattern: think about the creation of a private rds instance (test-instance-1): its endpoint will be available with the name test-instance-1.somerandomid.region.rds.amazonaws.com

Pros: centralised management, no need to maintain additional DNS servers.

Cons: information exposure about internal services and IP addresses, a standard naming convention has to be enforced and can be confusing: think about having a development version of the corporate website: should it be named www-dev.acme.com or dev-www.acme.com?

2. Use the same domain name with a "split DNS" configuration

Split DNS is a technique that enables you to return different values when a query matches a network or some conditions (using ACLS or views). You can return an internal IP address for www.acme.com for the on-premise network and the public website address for the entire internet.



As you can see, www.acme.com address would be different in the AWS Cloud VPC and on the internet, so internal services will be available only through the VPN connection with the proper web server configuration (using rules and ACLS).

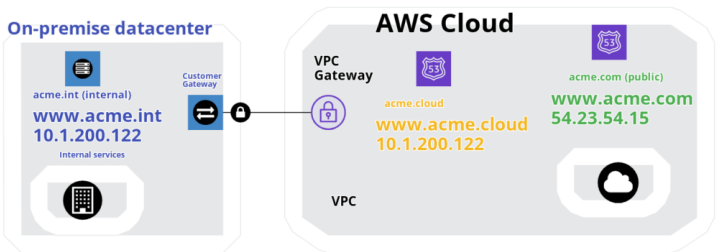
Pros: there will be a consistent naming when moving services between on-premise and the cloud: there's no need for application reconfiguration.

Cons: you need to maintain different DNS zones; it's easy to forget to add and maintain DNS records for every zone, sometimes leading to application problems (some services can be unreachable or wrongly configured).

3. Use different DNS zones for every environment

Different private DNS zones can help keep track of where services are.

Let's assume that we have a development environment on-premise and a testing environment on AWS. We'll use *acme.int* for our on-premise service and *acme.cloud* for the AWS Cloud.



We can also use different DNS subdomains, delegating the subzone to a DNS server that can maintain the configuration for the selected environment. In this scenario, our on-prem development environment will be on the *acme.int* DNS zone for our on-premise datacenter and *cloud.acme*.int for the testing environment on the AWS.

By creating an NS record in the acme.int zone that will refer to assigned DNS servers for *cloud.acme.int* we will be able to implement this scenario.

Pros: it's easy to identify a service environment.

Cons: refactoring and moving services to the cloud will need to reconfigure dependent applications to reflect the changing environment.

Resolving private zones hosted on Route53

When creating a private hosted zone on Route53, you can associate it with one or more VPCs. AWS automatically enables resolution for your EC2 instances and services by using an internal "magic" IP address that acts like a DNS server (it's the third reserved subnet address, so if your subnet is 192.168.0.0/24 it will be 192.168.1.2, if your subnet is 192.168.0.16/28 it will be 192.168.0.18).

You can reach the endpoint for DNS resolution only within a VPC, and if you try to use it forwarding queries from a VPN or a Direct Connect link, you'll find that it is not reachable.

We need to find a solution to implement our scenarios; let's see what AWS services can help us achieve our goal. Every implementation has its pros and cons; it's up to you to see what better fits your environment.

1. Use EC2 instances

You can deploy a pair of EC2 instances that will forward queries to the internal DNS zone, installing a DNS server of your choice (Bind, Microsoft DNS, Dnsmasq and everything you are comfortable using). Hint: If you are going to take the Advanced Networking Specialty exam, this is a common question!

You'll also need to configure DNS resolution to your on-premise environment using a conditional forwarder.

Pros: as long as you know to configure a DNS forwarding/caching only nameserver, it will take a little time for you to set up instances and install software

Cons: you also need to maintain this bit of infrastructure, regularly patch operating systems and software, schedule backups and implement a disaster recovery strategy

Costs: Assuming that a t3a.medium will be enough to handle your traffic, it will cost you 44$/month

2. Our little hack: use Simple AD

Even if you don't need an Active Directory environment, you can use AWS Simple AD. It will automatically forward DNS queries to Route53 private hosted zones and enable you to use it from an on-premise network.

In this scenario, you can configure, again, a conditional forwarder for your on-premise environment.

Pros: Simple AD is a fully managed service, so its service lifecycle is fully managed, including patching, backup, and high availability (it automatically deploys two domain controllers)

Cons: Since it's fully managed, you can't customise it (like enabling DNS hostname overrides using a text file for Dnsmasq)

Costs: A simple AD instance with a multi-AZ HA enabled by default will cost you 36$/month.

3. Use Route53 Resolver inbound and outbound endpoints

A Route53 resolver inbound endpoint deploys one or more VPC internal endpoints with a dedicated IP address reachable through VPN connections, Direct Connect or another peered VPC. This is the recommended implementation when dealing with complex network topologies. These endpoints will enable your servers to forward queries directly from an on-premise network.

If you need conditional forwarding to your on-premises environment, you'll have to configure an outbound endpoint. Il will create an Elastic Network Interface that can communicate with other on-prem DNS servers.

Pros: it's a fully managed service, highly configurable using only the AWS Console.

Cons: High pricing

Costs: You need at least two endpoints for a VPC; DNS queries billing is 0.40$/million queries. The price tag will be 183$/month.

These three configurations cover most implementations; you can mix and match a scenario and service implementation: subdomains and DNS delegation with Route53 resolver inbound endpoints, split DNS with Simple AD, different zones with EC2 instances.

To conclude

As you can see, there's always a tradeoff between ease of maintenance and service costs. There's no best implementation and DNS naming scheme: it's also a "Personal Taste" matter. Here at beSharp, you can find colleagues debating using a split DNS scheme or a subdomain delegation for new projects and even during coffee breaks!

In a hybrid cloud scenario, it's crucial to plan and implement the most appropriate architecture, choosing what better fits the environment without radically modifying a working solution.

What type of DNS architecture are you using?

Do you find yourselves arguing over using delegation, split DNS, or different zones? Let us know in the comments!

Damiano Giorgi

Ex on-prem systems engineer, lazy and prone to automating boring tasks. In constant search of technological innovations and new exciting things to experience. And that's why I love Cloud Computing! At this moment, the only "hardware" I regularly dedicate myself to is that my bass; if you can't find me in the office or in the band room try at the pub or at some airport, then!

Simone Merlini

CEO and co-founder of beSharp, Cloud Ninja and early adopter of any type of * aaS solution. I divide myself between the PC keyboard and the one with black and white keys; I specialize in deploying gargantuan dinners and testing vintage bottles.
