AWS Compute Blog

Integrating Amazon API Gateway private endpoints with on-premises networks

by Eric Johnson | on 09 JUL 2021 | in Amazon API Gateway, Amazon API Gateway, Networking & Content Delivery, Serverless | Permalink | Share
This post was written by Ahmed ElHaw, Sr. Solutions Architect

Using AWS Direct Connect or AWS Site-to-Site VPN, customers can establish a private virtual interface from their on-premises network directly to their Amazon Virtual Private Cloud (VPC). Hybrid networking enables customers to benefit from the scalability, elasticity, and ease of use of AWS services while using their corporate network.

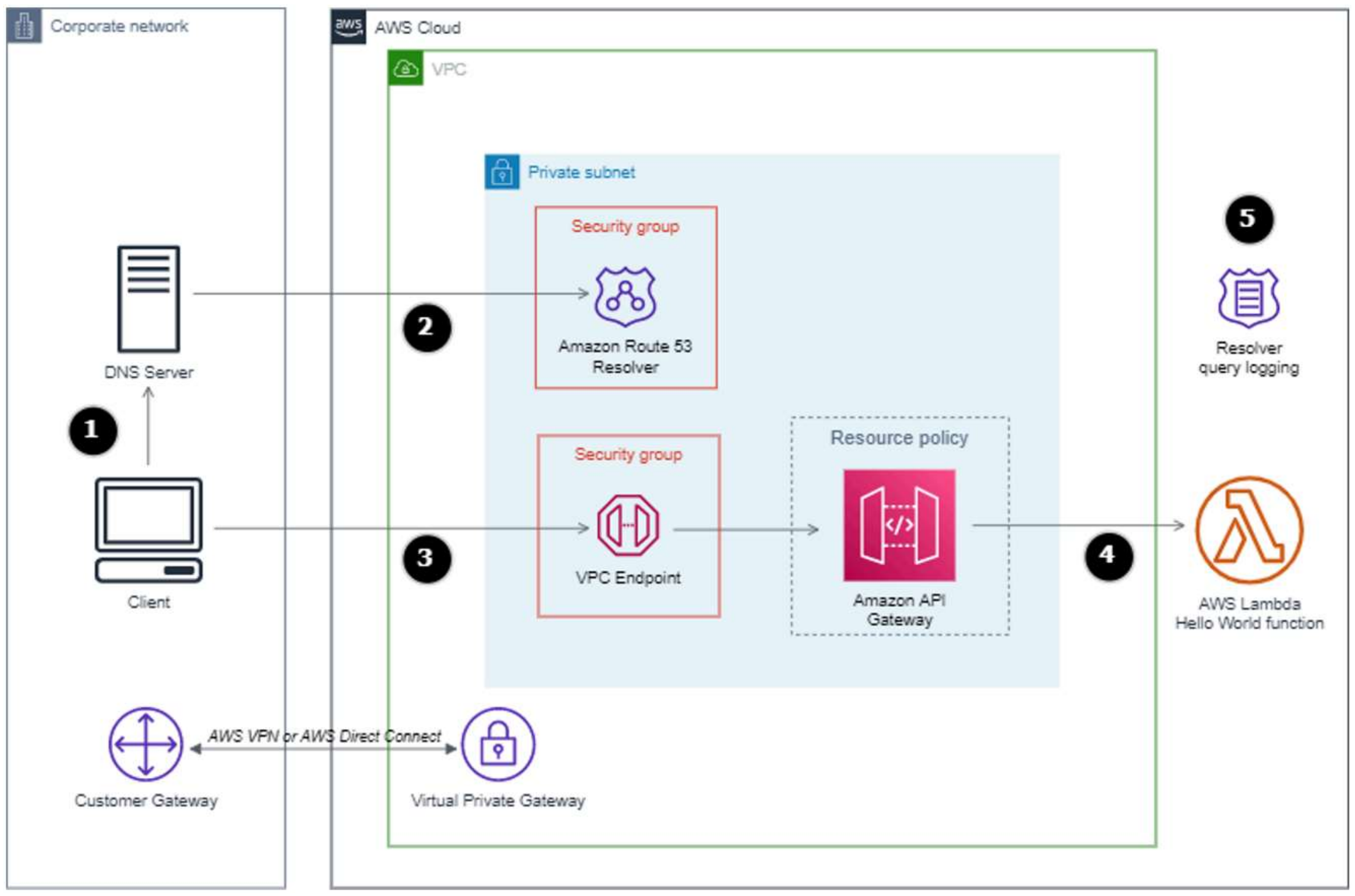
Amazon API Gateway can make it easier for developers to interface with and expose other services in a uniform and secure manner. You can use it to interface with other AWS services such as Amazon SageMaker endpoints for real-time machine learning predictions or serverless compute with AWS Lambda. API Gateway can also integrate with HTTP endpoints and VPC links in the backend.

This post shows how to set up a private API Gateway endpoint with a Lambda integration. It uses a Route 53 resolver, which enables on-premises clients to resolve AWS private DNS names.

Overview

API Gateway private endpoints allow you to use private API endpoints inside your VPC. When used with Route 53 resolver endpoints and hybrid connectivity, you can access APIs and their integrated backend services privately from on-premises clients.

You can deploy the example application using the AWS Serverless Application Model (AWS SAM). The deployment creates a private API Gateway endpoint with a Lambda integration and a Route 53 inbound endpoint. I explain the security configuration of the AWS resources used. This is the solution architecture:



Private API Gateway with a Hello World Lambda integration.

1. The client calls the private API endpoint (for example, GET https://abc123xyz0.execute-api.eu-west-1.amazonaws.com/demostage).
2. The client asks the on-premises DNS server to resolve (abc123xyz0.execute-api.eu-west-1.amazonaws.com). You must configure the on-premises DNS server to forward DNS queries for the AWS-hosted domains to the IP addresses of the inbound resolver endpoint. Refer to the documentation for your on-premises DNS server to configure DNS forwarders.
3. After the client successfully resolves the API Gateway private DNS name, it receives the private IP address of the VPC Endpoint of the API Gateway.
   Note: Call the DNS endpoint of the API Gateway for the HTTPS certificate to work. You cannot call the IP address of the endpoint directly.
4. Amazon API Gateway passes the payload to Lambda through an integration request.
5. If Route 53 Resolver query logging is configured, queries from on-premises resources that use the endpoint are logged.

Prerequisites

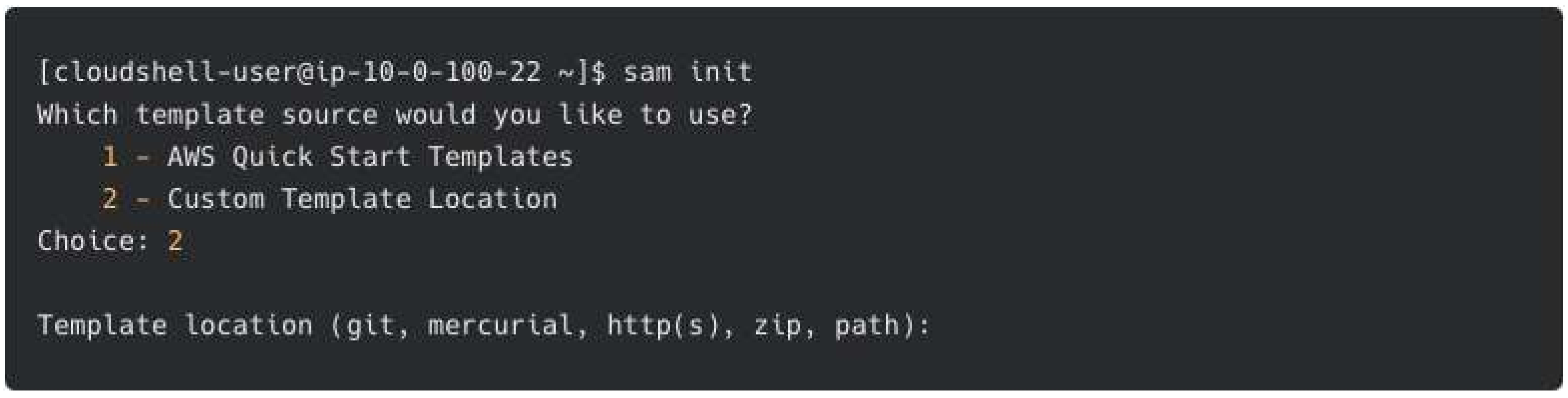
To deploy the example application in this blog post, you need:

- AWS credentials that provide the necessary permissions to create the resources. This example uses admin credentials.
- Amazon VPN or AWS Direct Connect with routing rules that allow DNS traffic to pass through to the Amazon VPC.

- The AWS SAM CLI installed.
- Clone the GitHub repository.

Deploying with AWS SAM

1. Navigate to the cloned repo directory. Alternatively, use the sam init command and paste the repo URL:



```
[cloudshell-user@ip-10-0-100-22 ~]$ sam init
Which template source would you like to use?
    1 - AWS Quick Start Templates
    2 - Custom Template Location
Choice: 2

Template location (git, mercurial, http(s), zip, path):
```

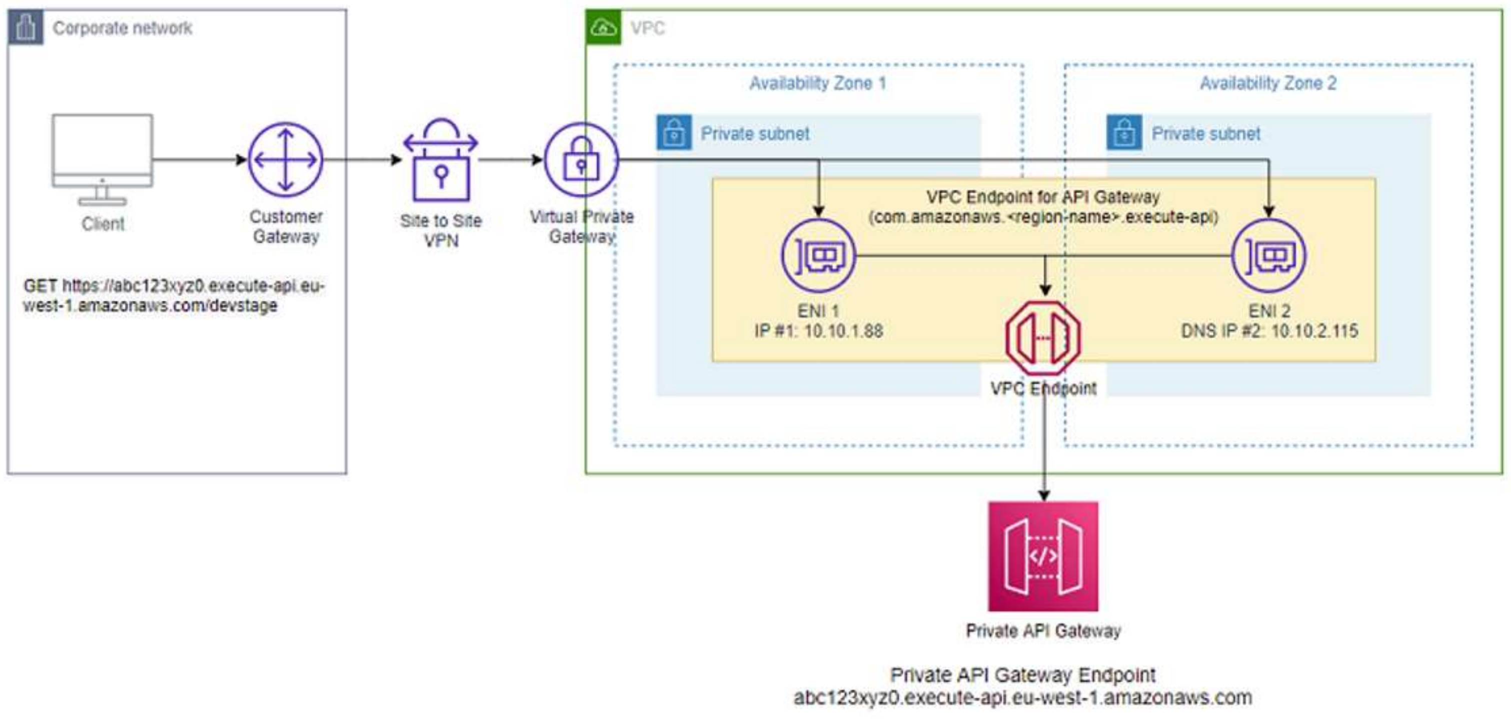
SAM init example

2. Build the AWS SAM application:
   `sam build`
3. Deploy the AWS SAM application:
   `sam deploy –guided`

This stack creates and configures a virtual private cloud (VPC) configured with two private subnets (for resiliency) and DNS resolution enabled. It also creates a VPC endpoint with (service name = “com.amazonaws.{region}.execute-api”), Private DNS Name = enabled, and a security group set to allow TCP Port 443 inbound from a managed prefix list. You can edit the created prefix list with one or more CIDR block(s).

It also deploys an API Gateway private endpoint and an API Gateway resource policy that restricts access to the API, except from the VPC endpoint. There is also a “Hello world” Lambda function and a Route 53 inbound resolver with a security group that allows TCP/UDP DNS port inbound from the on-premises prefix list.

A VPC endpoint is a logical construct consisting of elastic network interfaces deployed in subnets. The elastic network interface is assigned a private IP address from your subnet space. For high availability, deploy in at least two Availability Zones.


Corporate network
Client
Customer Gateway
GET https://abc123xyz0.execute-api.eu-west-1.amazonaws.com/devstage
Site to Site VPN
Virtual Private Gateway
VPC
Availability Zone 1
Availability Zone 2
Private subnet
Private subnet
VPC Endpoint for API Gateway (com.amazonaws.<region-name>.execute-api)
ENI 1 IP #1: 10.10.1.88
ENI 2 DNS IP #2: 10.10.2.115
VPC Endpoint
Private API Gateway
Private API Gateway Endpoint abc123xyz0.execute-api.eu-west-1.amazonaws.com

Private API Gateway VPC endpoint

Route 53 inbound resolver endpoint

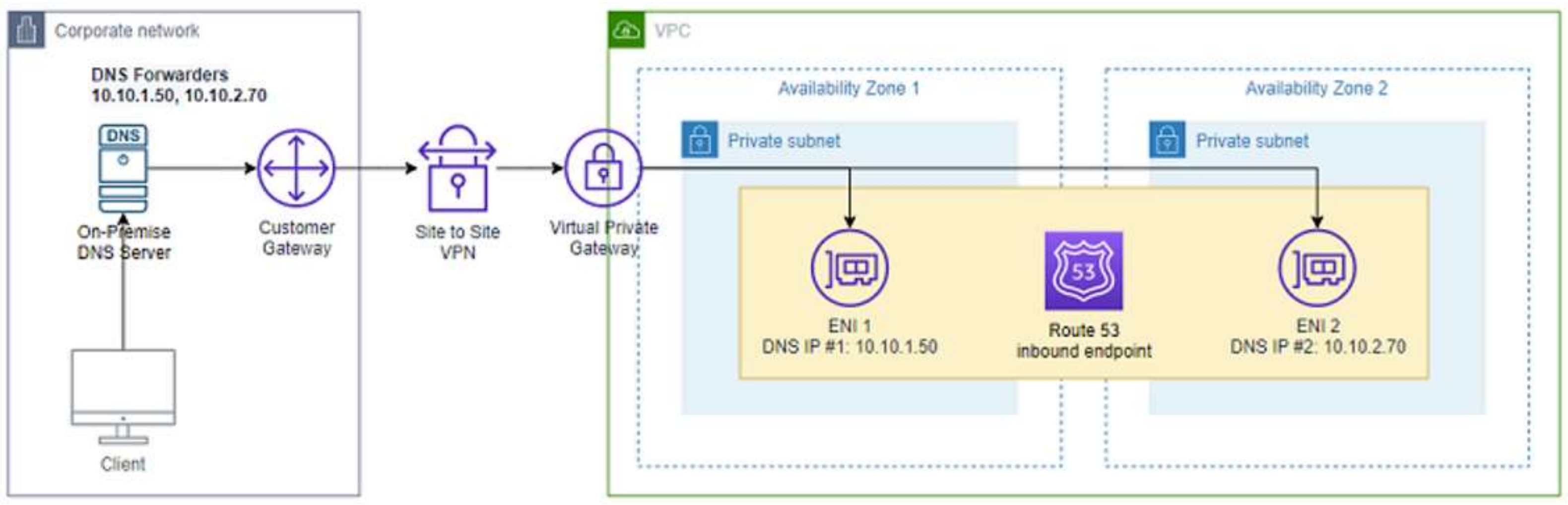
Route 53 resolver is the Amazon DNS server. It is sometimes referred to as “AmazonProvidedDNS” or the “.2 resolver” that is available by default in all VPCs. Route 53 resolver responds to DNS queries from AWS resources within a VPC for public DNS records, VPC-specific DNS names, and Route 53 private hosted zones.

Integrating your on-premises DNS server with AWS DNS server requires a Route 53 resolver inbound endpoint (for DNS queries that you’re forwarding to your VPCs). When creating an API Gateway private endpoint, a private DNS name is created by API Gateway. This endpoint is resolved automatically from within your VPC.

However, the on-premises servers learn about this hostname from AWS. For this, create a Route 53 inbound resolver endpoint and point your on-premises DNS server to it. This allows your corporate network resources to resolve AWS private DNS hostnames.

To improve reliability, the resolver requires that you specify two IP addresses for DNS queries. AWS recommends configuring IP addresses in two different Availability Zones. After you add the first two IP addresses, you can optionally add more in the same or different Availability Zone.

The inbound resolver is a logical resource consisting of two elastic network interfaces. These are deployed in two different Availability Zones for resiliency.



Route 53 inbound resolver

Configuring the security groups and resource policy

In the security pillar of the AWS Well-Architected Framework, one of the seven design principles is applying security at all layers: Apply a defense in depth approach with multiple security controls. Apply to all layers (edge of network, VPC, load balancing, every instance and compute service, operating system, application, and code).

A few security configurations are required for the solution to function:

- The resolver security group (referred to as 'ResolverSG' in solution diagram) inbound rules must allow TCP and UDP on port 53 (DNS) from your on-premises network-managed prefix list (source). *Note: configure the created managed prefix list with your on-premises network CIDR blocks.*
- The security group of the VPC endpoint of the API Gateway "VPCEndpointSG" must allow HTTPS access from your on-premises network-managed prefix list (source). *Note: configure the created managed prefix list with your on-premises network CIDR blocks.*
- For a private API Gateway to work, a resource policy must be configured. The AWS SAM deployment sets up an API Gateway resource policy that allows access to your API from the VPC endpoint. We are telling API Gateway to deny any request explicitly unless it is originating from a defined source VPC endpoint.
  Note: AWS SAM template creates the following policy:

  JSON
  ```json
  {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:eu-west-1:12345678901:dligg9dxuk/DemoStage/GET/hello"
        },
        {
            "Effect": "Deny",
            "Principal": "*",
            "Action": "execute-api:Invoke",
            "Resource": "arn:aws:execute-api:eu-west-1: 12345678901:dligg9dxuk/DemoStage/GET/hello",
            "Condition": {
                "StringNotEquals": {
                    "aws:SourceVpce": "vpce-0ac4147ba9386c9z7"
                }
            }
  ```

The AWS SAM deployment creates a Hello World Lambda. For demonstration purposes, the Lambda function always returns a successful response, conforming with API Gateway integration response.

Testing the solution

To test, invoke the API using a curl command from an on-premises client. To get the API URL, copy it from the on-screen AWS SAM deployment outputs. Alternatively, from the console go to AWS CloudFormation outputs section.

CloudFormation outputs

Next, go to Route 53 resolvers, select the created inbound endpoint and note of the endpoint IP addresses. Configure your on-premises DNS forwarder with the IP addresses. Refer to the documentation for your on-premises DNS server to configure DNS forwarders.

Route 53 resolver IP addresses

Finally, log on to your on-premises client and call the API Gateway endpoint. You should get a success response from the API Gateway as shown.

Bash
```
curl https://dligg9dxuk.execute-api.eu-west-1.amazonaws.com/DemoStage/hello

{"response": {"resultStatus": "SUCCESS"}}
```

Monitoring and troubleshooting

Route 53 resolver query logging allows you to log the DNS queries that originate in your VPCs. It shows which domain names are queried, the originating AWS resources (including source IP and instance ID) and the responses.

You can log the DNS queries that originate in VPCs that you specify, in addition to the responses to those DNS queries. You can also log DNS queries from on-premises resources that use an inbound resolver endpoint, and DNS queries that use an outbound resolver endpoint for recursive DNS resolution.

After configuring query logging from the console, you can use Amazon CloudWatch as the destination for the query logs. You can use this feature to view and troubleshoot the resolver.

JSON
```
{
    "version": "1.100000",
    "account_id": "1234567890123",
    "region": "eu-west-1",
    "vpc_id": "vpc-0c00ca6aa29c8472f",
    "query_timestamp": "2021-04-25T12:37:34Z",
    "query_name": "dligg9dxuk.execute-api.eu-west-1.amazonaws.com.",
    "query_type": "A",
    "query_class": "IN",
    "rcode": "NOERROR",
    "answers": [
        {
            "Rdata": "10.0.140.226”, API Gateway VPC Endpoint IP#1
            "Type": "A",
            "Class": "IN"
        },
        {
            "Rdata": "10.0.12.179", API Gateway VPC Endpoint IP#2
            "Type": "A",
```

Cleaning up

To remove the example application, navigate to CloudFormation and delete the stack.

Conclusion

API Gateway private endpoints allow use cases for building private API–based services inside your VPCs. You can keep both the frontend to your application (API Gateway) and the backend service private inside your VPC.

I discuss how to access your private APIs from your corporate network through Direct Connect or Site-to-Site VPN without exposing your endpoints to the internet. You deploy the demo using AWS Serverless Application Model (AWS SAM). You can also change the template for your own needs.

To learn more, visit the API Gateway tutorials and workshops page in the API Gateway developer guide.

For more serverless learning resources, visit Serverless Land.
TAGS: contributed, serverless