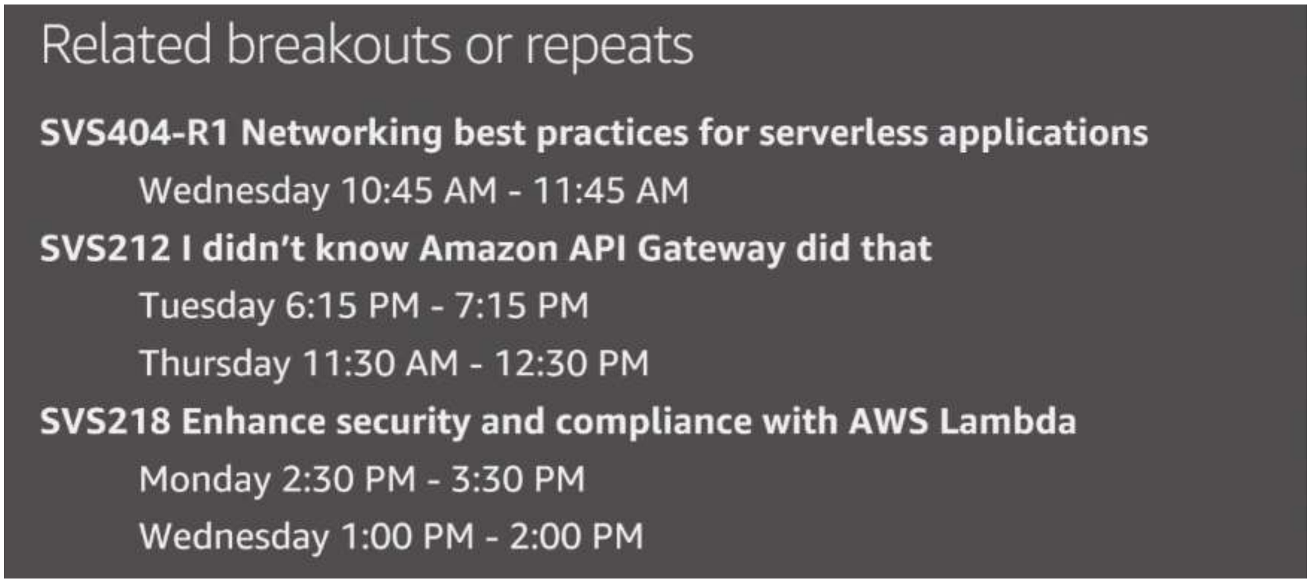
SVS404-R

# Networking best practices for serverless applications

Nicki Stone
Senior Software Engineer
Amazon Web Services

re:Invent

Serverless technologies such as AWS Lambda have removed the burden of server management, but what about networking? When should you put a Lambda in an Amazon VPC? How do you balance security vs. the flexibility offered by Lambda? What are the best practices for working with private endpoints, NATs, and peering? In this session, we go over the best practices of working with Lambda functions from a networking perspective. We talk about how networking impacts performance and cost and how to make sure that your network design allows for scale and meeting strict security concerns.

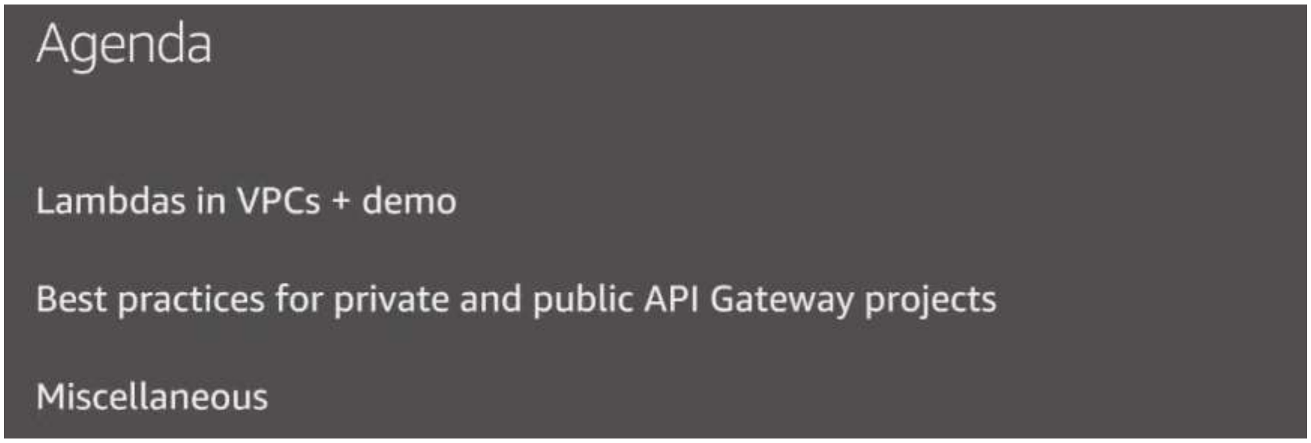


## Related breakouts or repeats

**SVS404-R1 Networking best practices for serverless applications**
Wednesday 10:45 AM - 11:45 AM

**SVS212 I didn't know Amazon API Gateway did that**
Tuesday 6:15 PM - 7:15 PM
Thursday 11:30 AM - 12:30 PM

**SVS218 Enhance security and compliance with AWS Lambda**
Monday 2:30 PM - 3:30 PM
Wednesday 1:00 PM - 2:00 PM



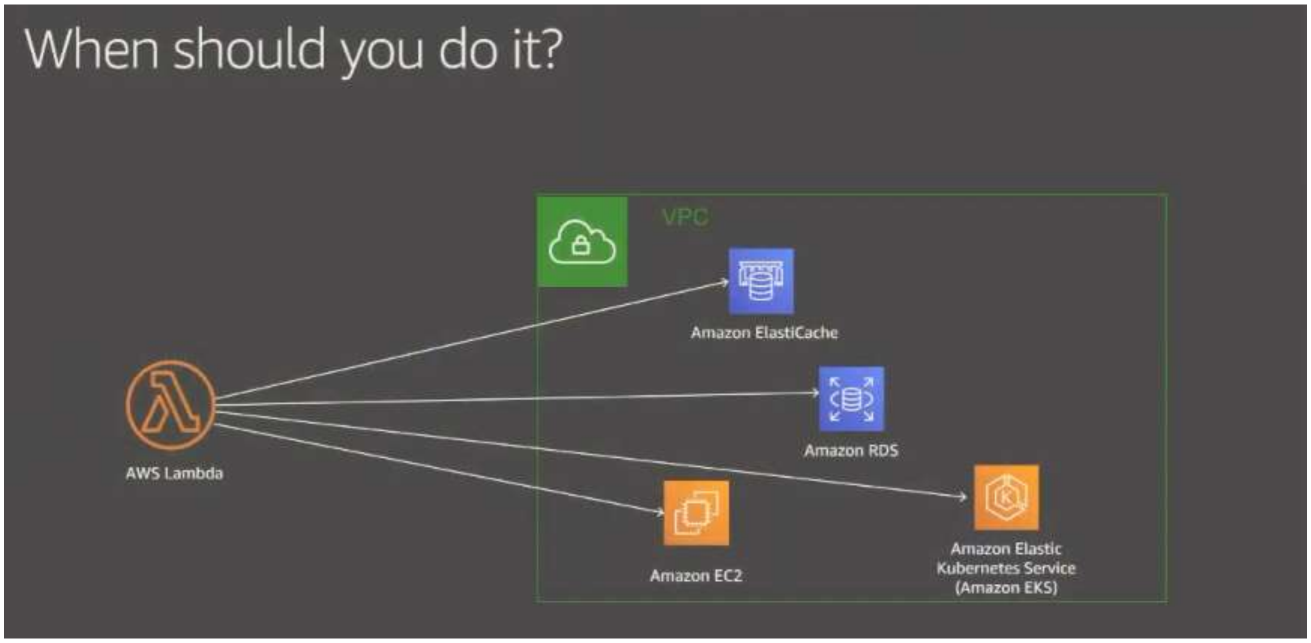
## Agenda

Lambdas in VPCs + demo

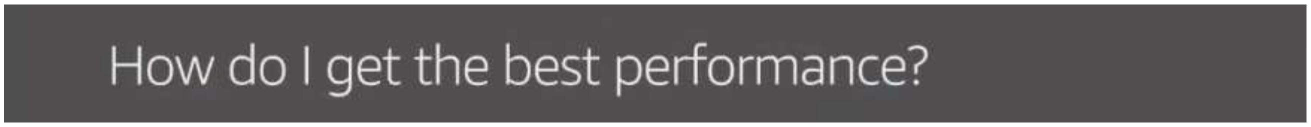
Best practices for private and public API Gateway projects

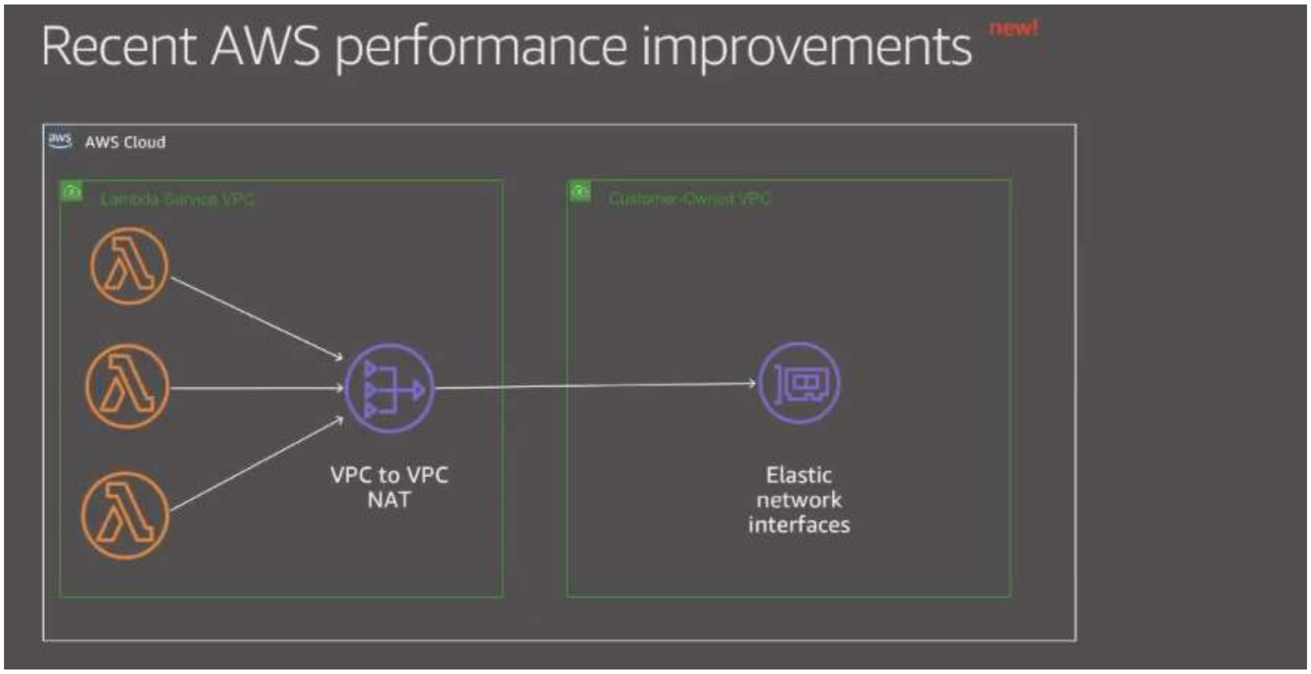
Miscellaneous

## Lambda & VPC

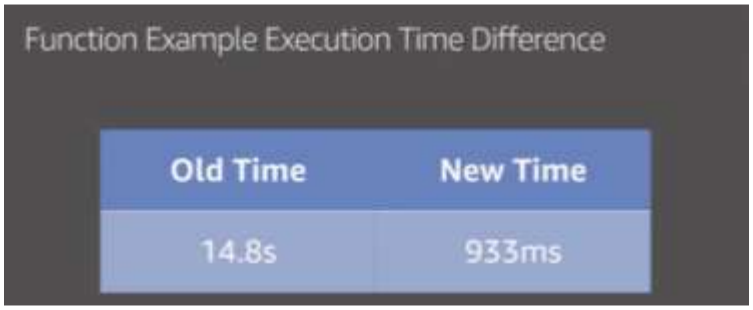

### When should you do it?

Only do it when a lambda needs to talk to something in your VPC. The lambda is in its own VPC and you are making a connection/attachment from that VPC to your VPC.


### How do I get the best performance?

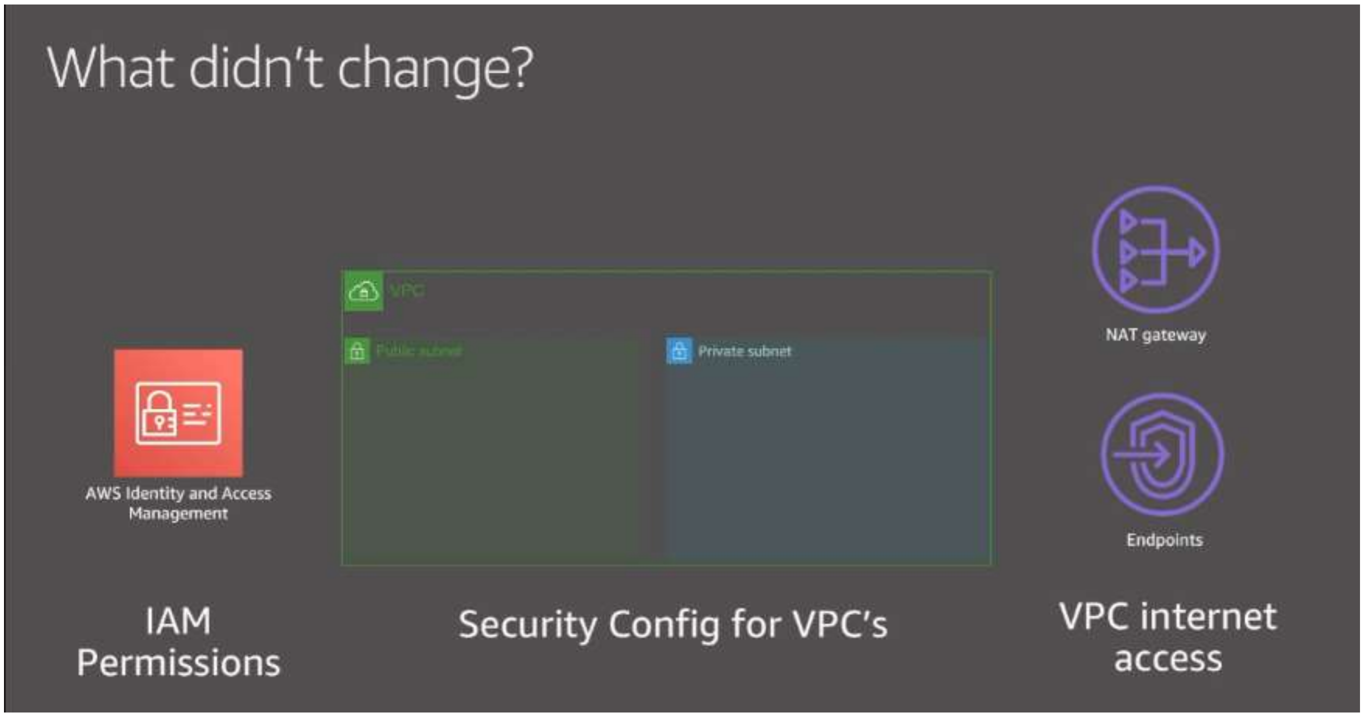

### Recent AWS performance improvements new!

Previously when you attach a Lambda to your VPC and you execute it, an ENI is created for you to create a network tunnel to your VPC for the lambda to use. The new way basically maps your VPC to a hyper-plane ENI and then to an ENI in your account, this means that the ENI in your account is created at the time of the lambda creation or when you add VPC settings to the lambda. This creates a huge performance gain because the tunnel can be created immediately you attach the lambda. This ENI can now be shared across multiple unique SG/subnet combination across function in this image.



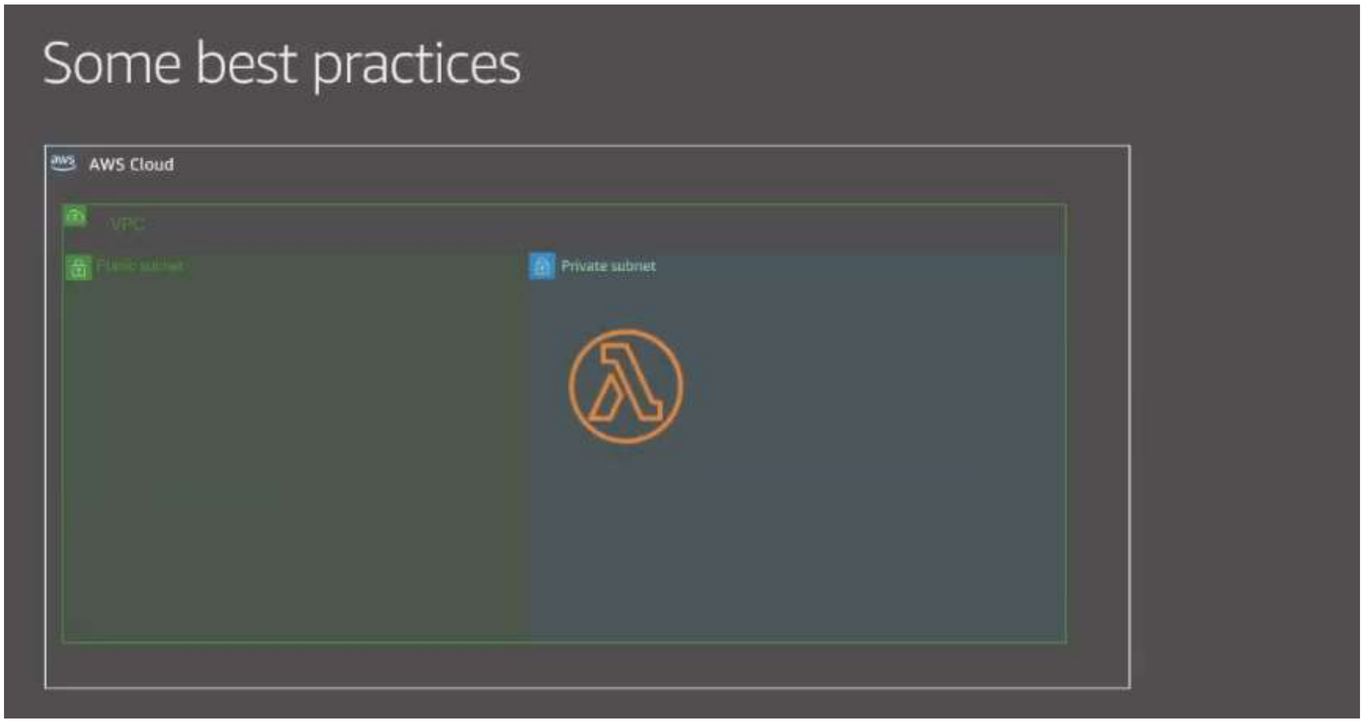
Function Example Execution Time Difference

| Old Time | New Time |
|---|---|
| 14.8s | 933ms |



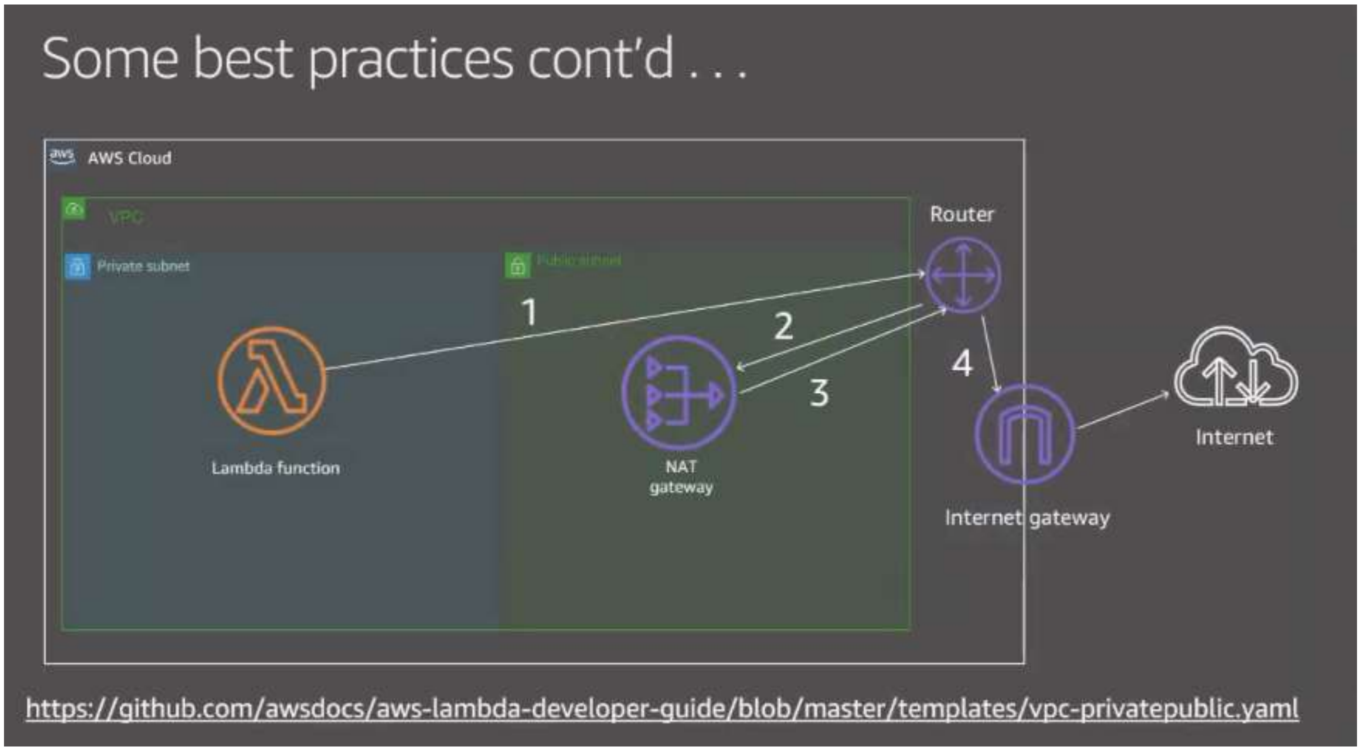
## What didn't change?

IAM Permissions did not change and you still need to have a role attached to your lambda function that can create and delete ENIs. You still control the security config and security groups of the ENIs and continue to apply normal security controls and VPC configurations. You still need a NAT gateway device to give your function internet access or you can use VPC endpoints to access services outside your VPC.



## Some best practices

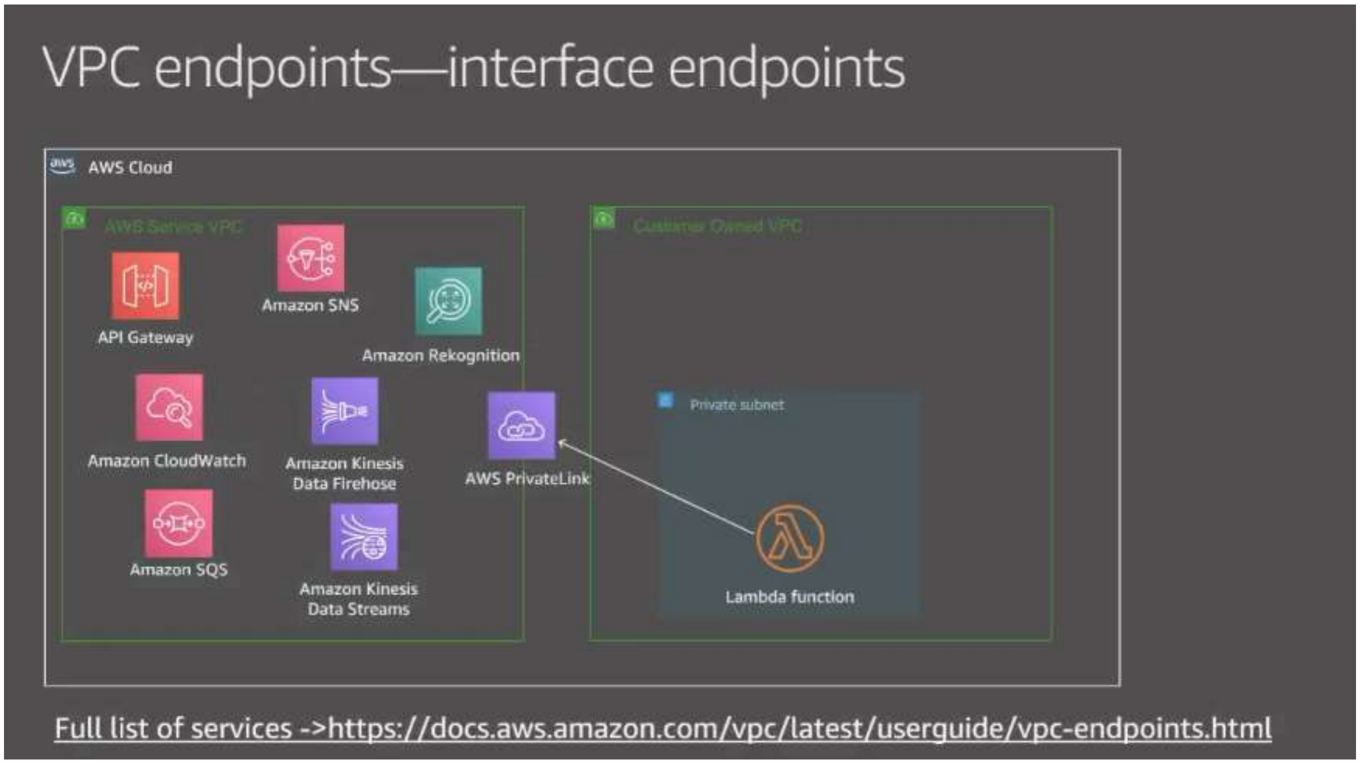
If your function does not talk to anything on the internet then you should put it inside a Private subnet.



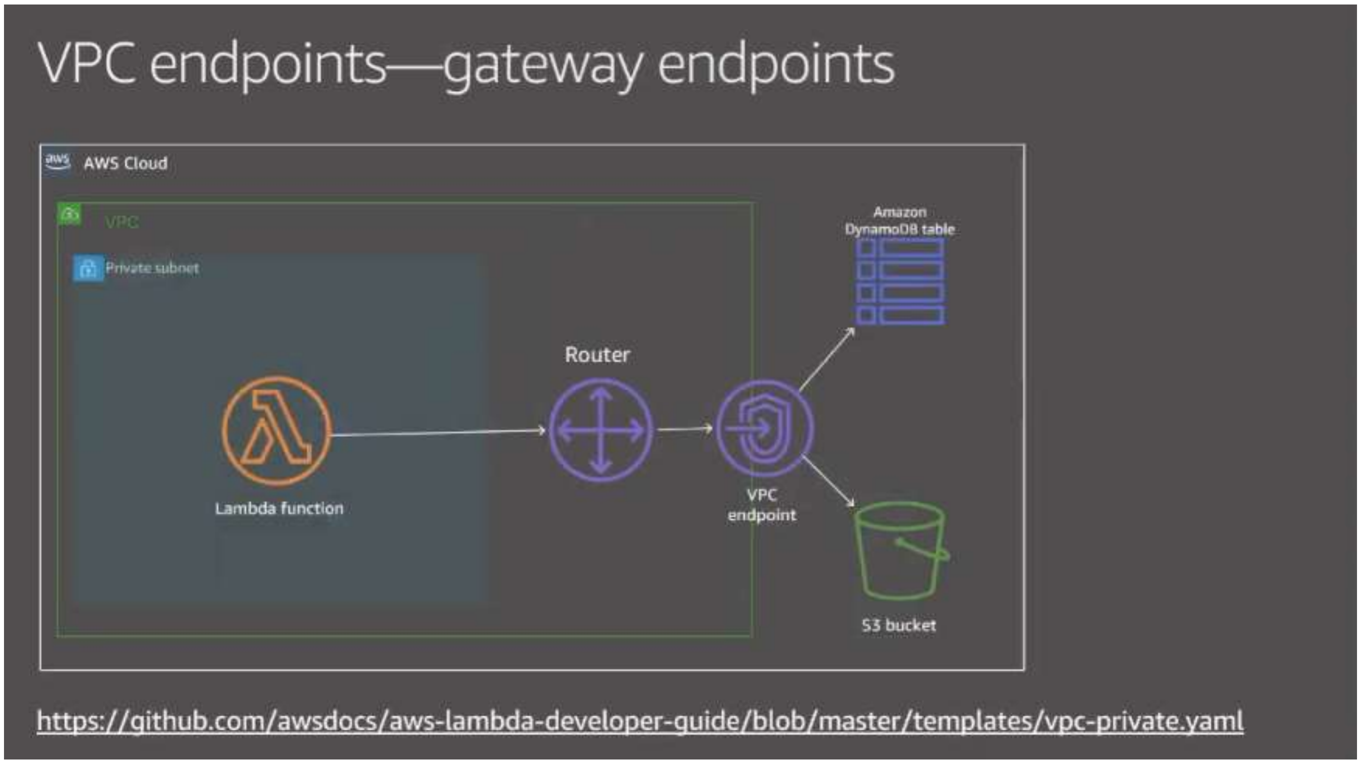
## Some best practices cont'd . . .

https://github.com/awsdocs/aws-lambda-developer-guide/blob/master/templates/vpc-privatepublic.yaml

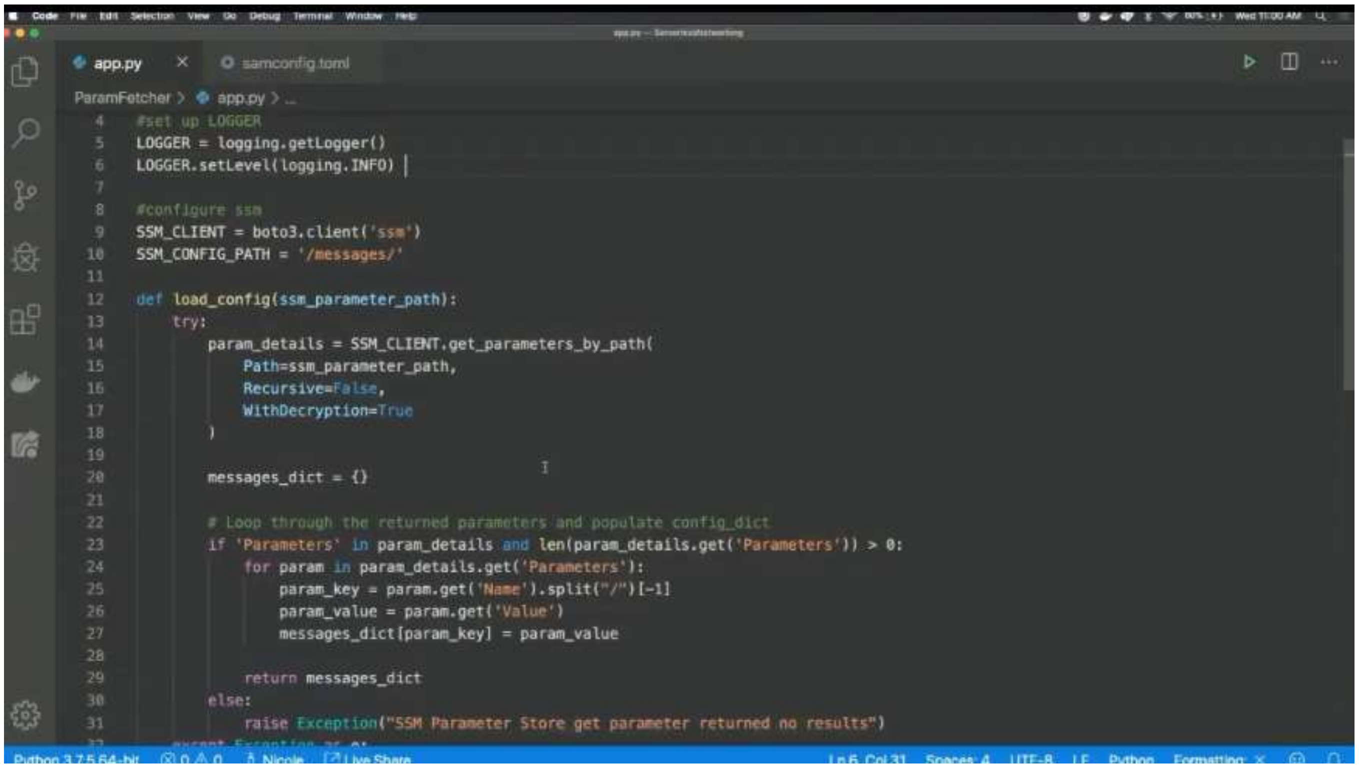
Even if your function needs internet access, you should still put your function in a Private subnet and use a NAT gateway to provide it access to the internet.


VPC endpoints—interface endpoints
AWS Cloud
AWS Service VPC
Customer Owned VPC
API Gateway
Amazon SNS
Amazon Rekognition
Amazon CloudWatch
Amazon Kinesis Data Firehose
AWS PrivateLink
Amazon SQS
Amazon Kinesis Data Streams
Private subnet
Lambda function
Full list of services ->https://docs.aws.amazon.com/vpc/latest/userguide/vpc-endpoints.html

***VPC endpoints*** are special endpoints from AWS that enables you to privately connect your VPC to some supported AWS services without needing an internet gateway connection, a NAT device, a VPN connection, a DirectConnect connection or others. Several services offer ***VPC endpoints of 2 kinds***, an ***interface*** (an elastic network interface hosted here by AWS PrivateLink that gives a private IP address from the IP address range of your subnet that will serve as an entry point for the service destined to that service in the AWS Service VPC) or a **gateway**.

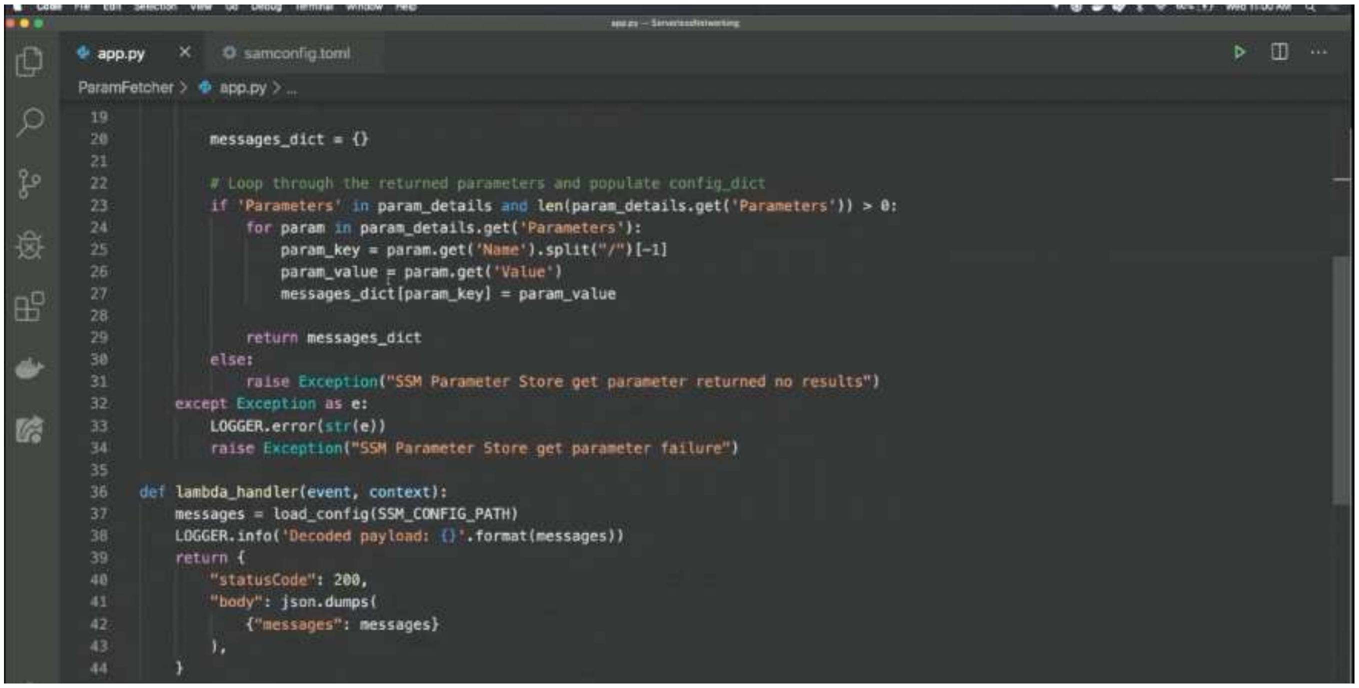

VPC endpoints—gateway endpoints
AWS Cloud
VPC
Private subnet
Lambda function
Router
VPC endpoint
Amazon DynamoDB table
S3 bucket
https://github.com/awsdocs/aws-lambda-developer-guide/blob/master/templates/vpc-private.yaml

A VPC Endpoint Gateway is a gateway that you specify as a target for a route in your route table for traffic destined for AWS Services that supports gateway VPC Endpoints, which are DynamoDB and S3.



```python
#set up LOGGER
LOGGER = logging.getLogger()
LOGGER.setLevel(logging.INFO)

#configure ssm
SSM_CLIENT = boto3.client('ssm')
SSM_CONFIG_PATH = '/messages/'

def load_config(ssm_parameter_path):
    try:
        param_details = SSM_CLIENT.get_parameters_by_path(
            Path=ssm_parameter_path,
            Recursive=False,
            WithDecryption=True
        )

        messages_dict = {}

        # Loop through the returned parameters and populate config_dict
        if 'Parameters' in param_details and len(param_details.get('Parameters')) > 0:
            for param in param_details.get('Parameters'):
                param_key = param.get('Name').split("/")[-1]
                param_value = param.get('Value')
                messages_dict[param_key] = param_value

            return messages_dict
        else:
            raise Exception("SSM Parameter Store get parameter returned no results")
```

We have a Lambda function that is hitting the AWS Parameter Store to grab a parameter under the **/messages/** route and just print it out

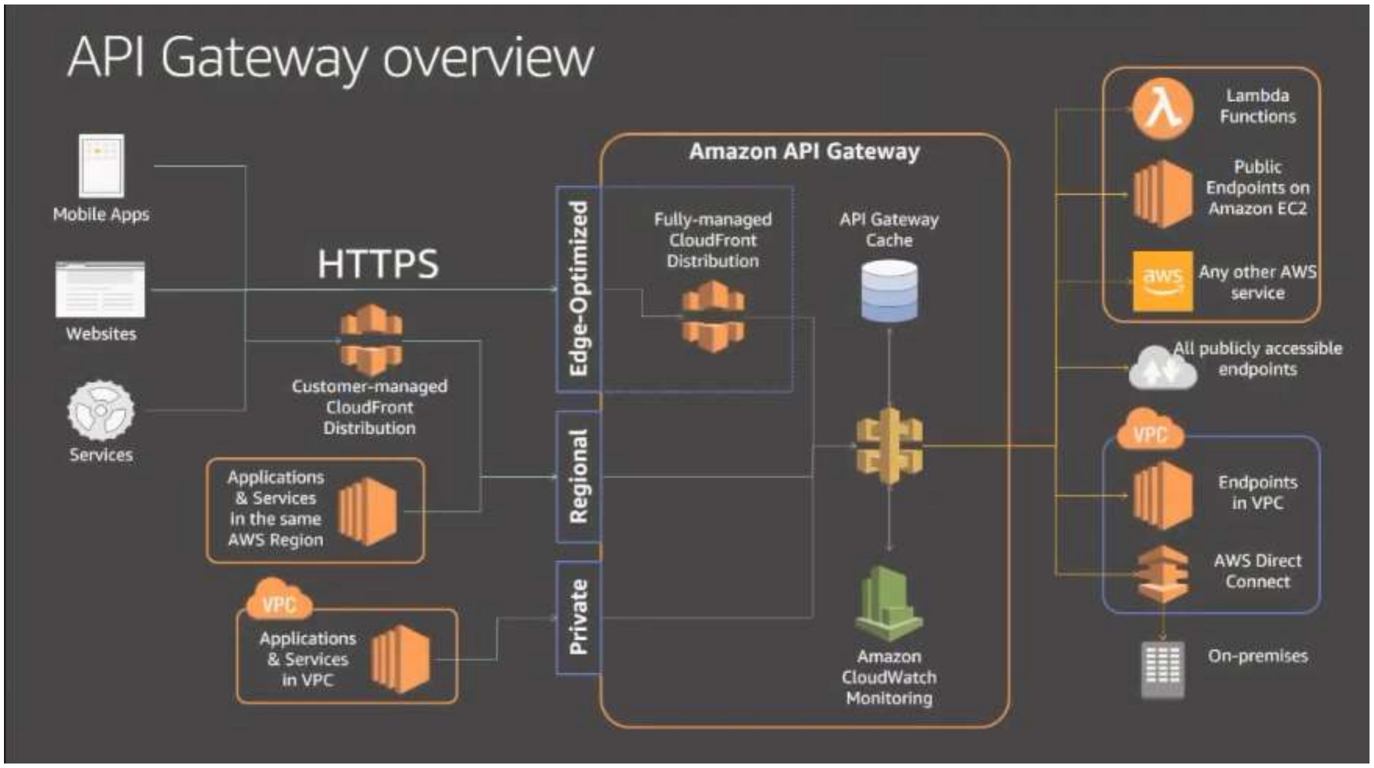


```python

        messages_dict = {}

        # Loop through the returned parameters and populate config_dict
        if 'Parameters' in param_details and len(param_details.get('Parameters')) > 0:
            for param in param_details.get('Parameters'):
                param_key = param.get('Name').split("/")[-1]
                param_value = param.get('Value')
                messages_dict[param_key] = param_value

            return messages_dict
        else:
            raise Exception("SSM Parameter Store get parameter returned no results")
    except Exception as e:
        LOGGER.error(str(e))
        raise Exception("SSM Parameter Store get parameter failure")

def lambda_handler(event, context):
    messages = load_config(SSM_CONFIG_PATH)
    LOGGER.info('Decoded payload: {}'.format(messages))
    return {
        "statusCode": 200,
        "body": json.dumps(
            {"messages": messages}
        ),
    }
```

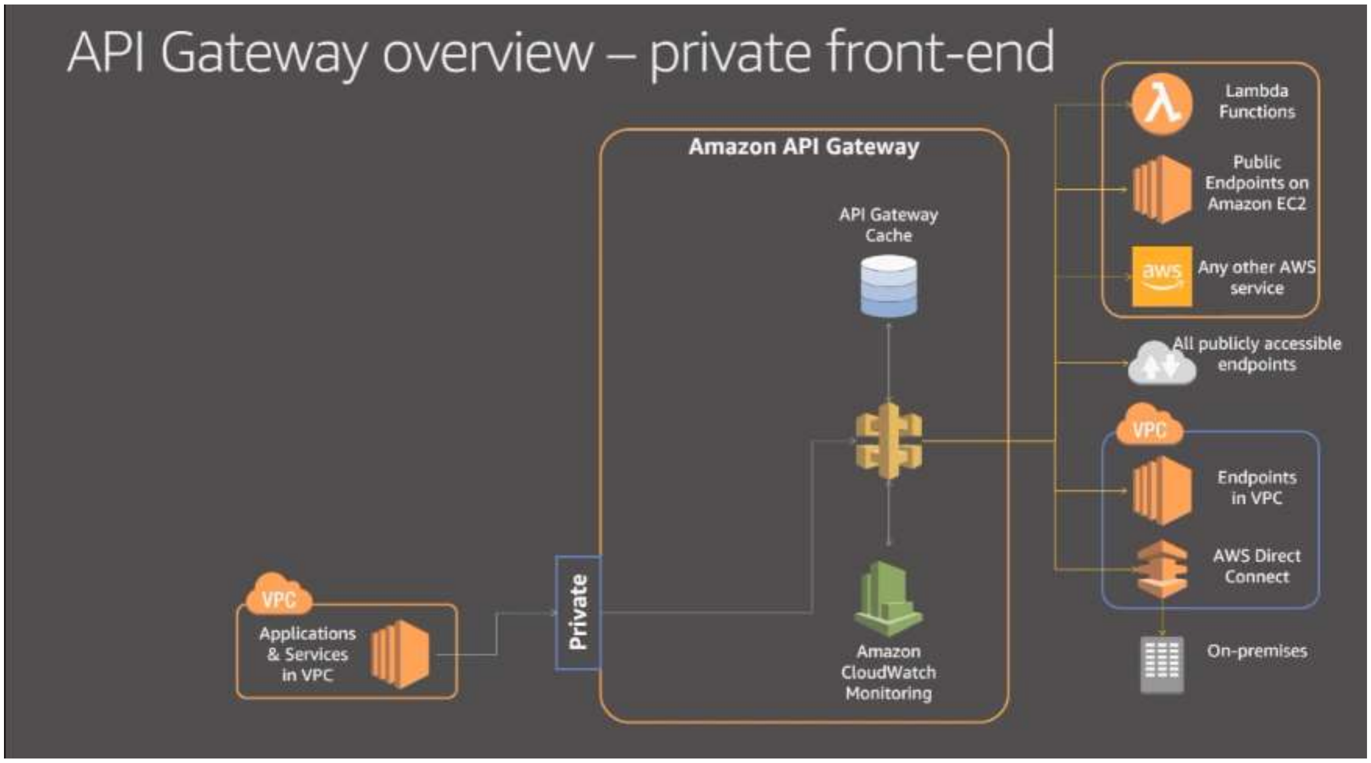
We have deployed the lambda and attached it to a VPC. We also created a VPC endpoint interface for getting the data from the Parameter Store.

## Private API Gateway best practices

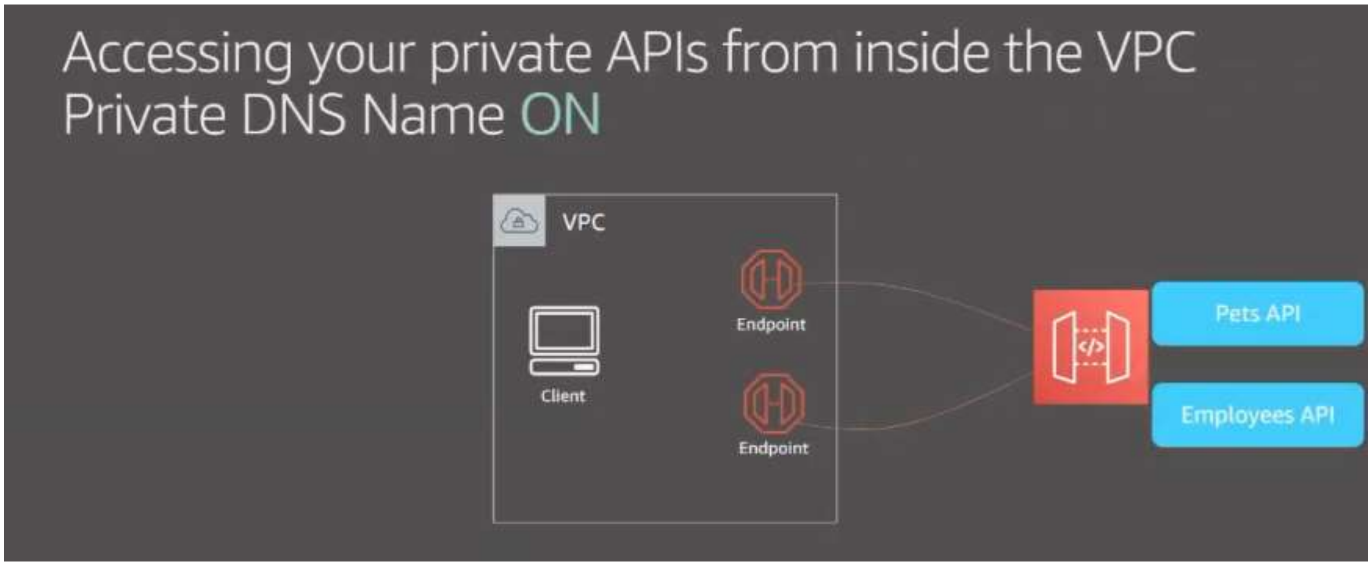
Setting up a private API Gateway can be difficult as we see below.


API Gateway overview
Mobile Apps
Websites
Services
HTTPS
Customer-managed CloudFront Distribution
Applications & Services in the same AWS Region
VPC
Applications & Services in VPC
Edge-Optimized
Regional
Private
Amazon API Gateway
Fully-managed CloudFront Distribution
API Gateway Cache
Amazon CloudWatch Monitoring
Lambda Functions
Public Endpoints on Amazon EC2
aws
Any other AWS service
All publicly accessible endpoints
VPC
Endpoints in VPC
AWS Direct Connect
On-premises

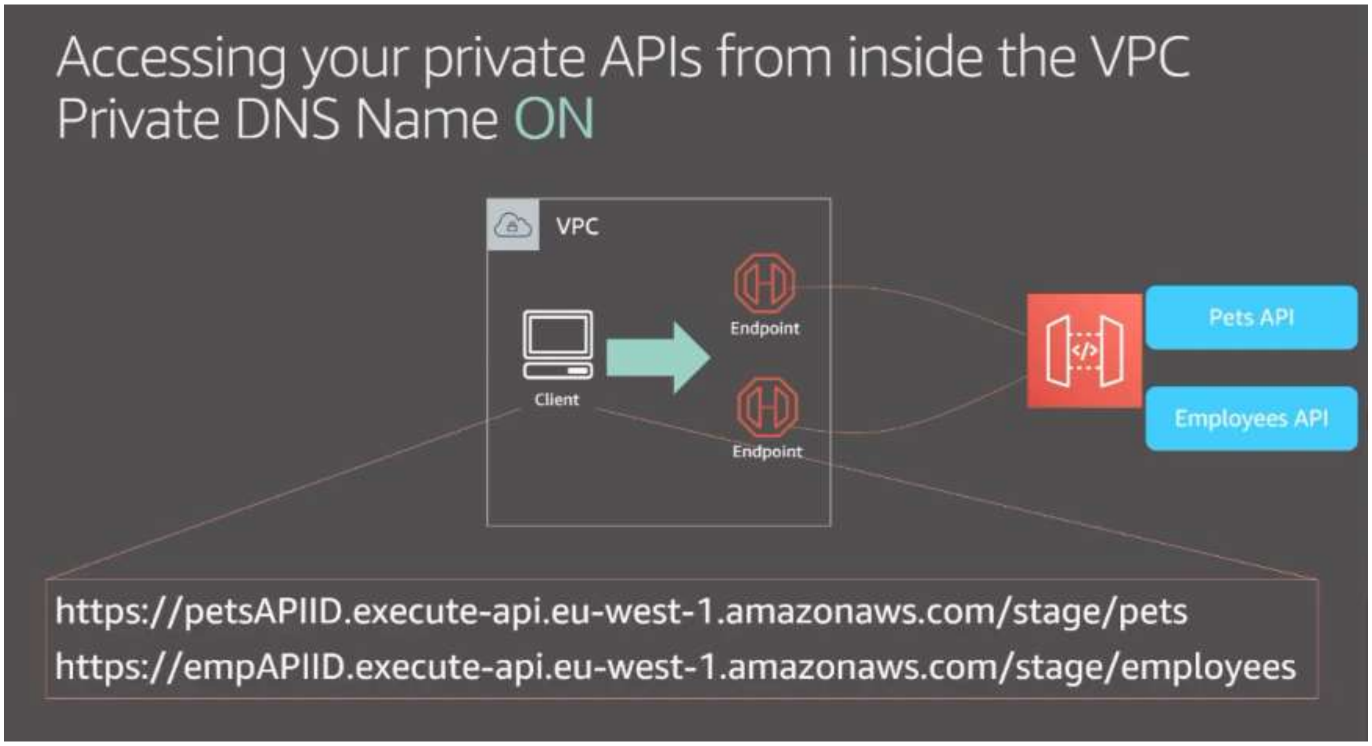
These is an overview of the different endpoints that we can create with an API Gateway, Private, Regional, and Edge-optimized.


API Gateway overview – private front-end
Amazon API Gateway
API Gateway Cache
Amazon CloudWatch Monitoring
Private
VPC
Applications & Services in VPC
Lambda Functions
Public Endpoints on Amazon EC2
Any other AWS service
All publicly accessible endpoints
VPC
Endpoints in VPC
AWS Direct Connect
On-premises

The actual frontend of the Private API Gateway is a VPC and not a device or web app, it is a VPC that needs to hit the Private API Gateway endpoint.

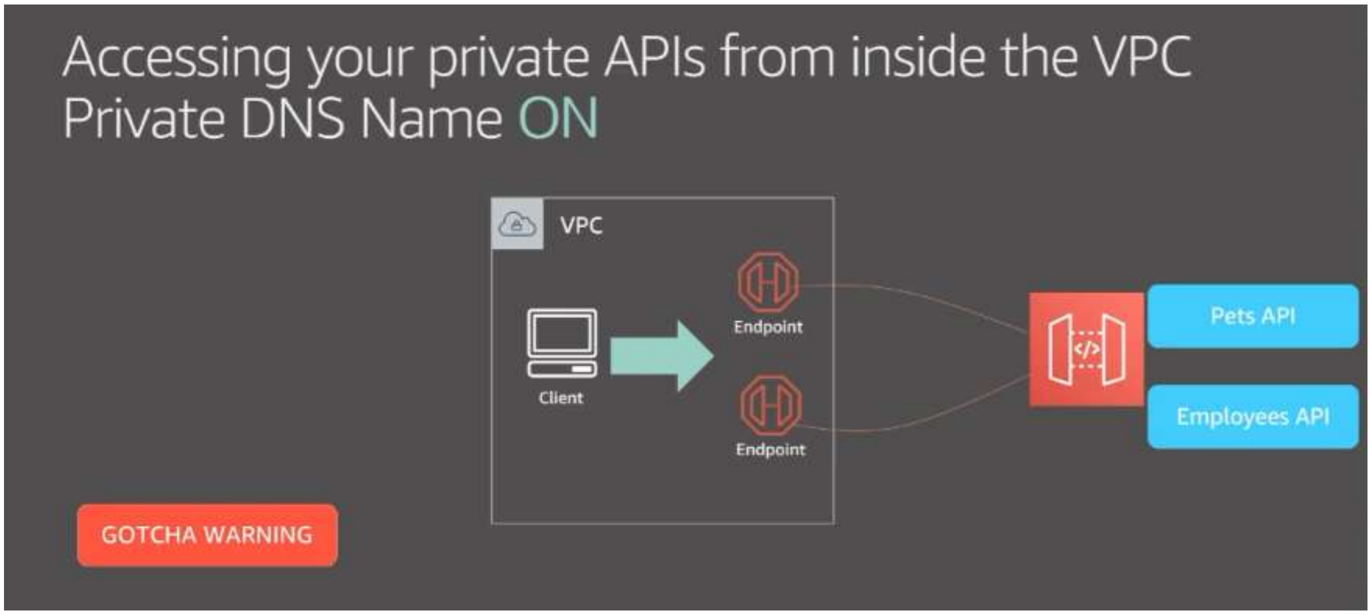


In order to access our private API Gateway endpoint, we need to create VPC endpoints like above, but we need to tell those endpoints that they belong to the Private API Gateway.

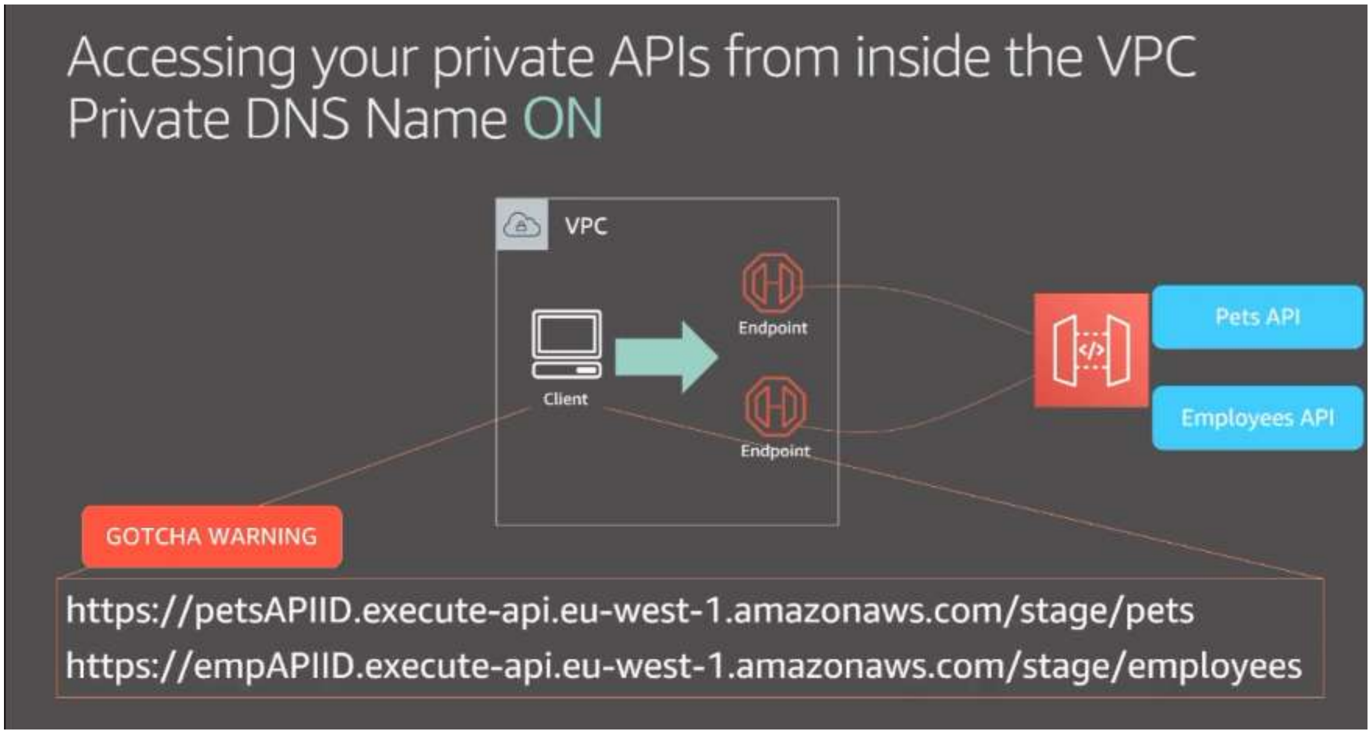


```
https://petsAPIID.execute-api.eu-west-1.amazonaws.com/stage/pets
https://empAPIID.execute-api.eu-west-1.amazonaws.com/stage/employees
```

We can use DNS for this so that we can call our private API Gateway endpoints like we would normally do as above.

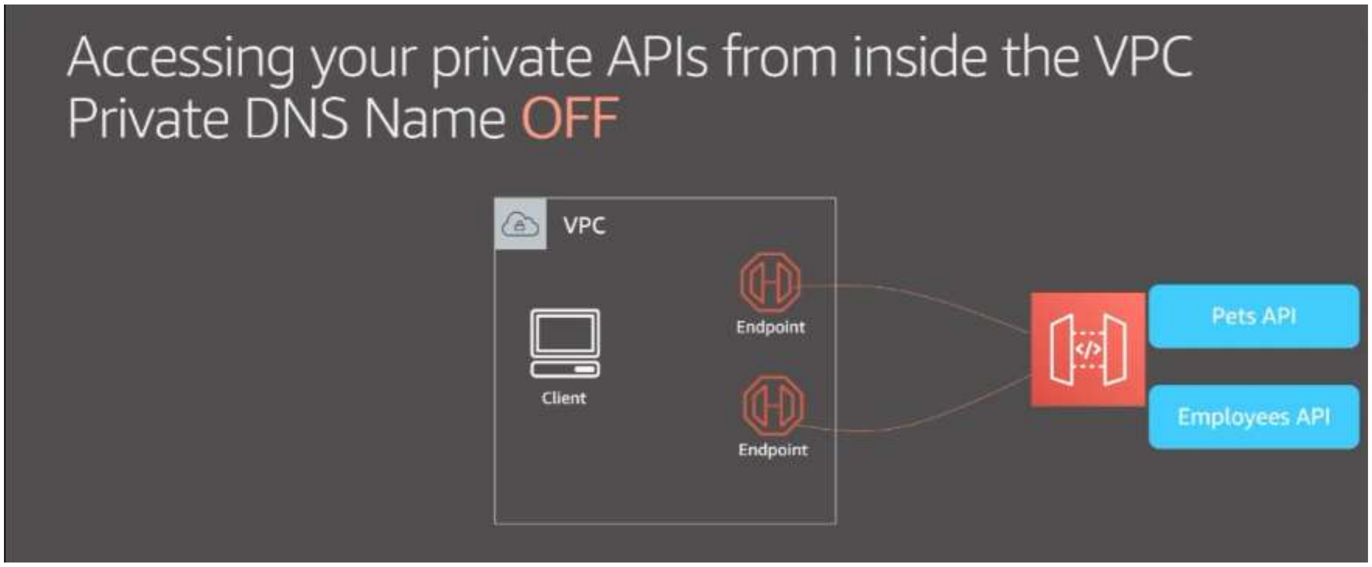


But what if we need to turn off DNS? We might turn it off in situations when our VPC needs to talk to multiple different kinds of API Gateways like public and private. We don’t all that traffic redirected to those VPC endpoints.



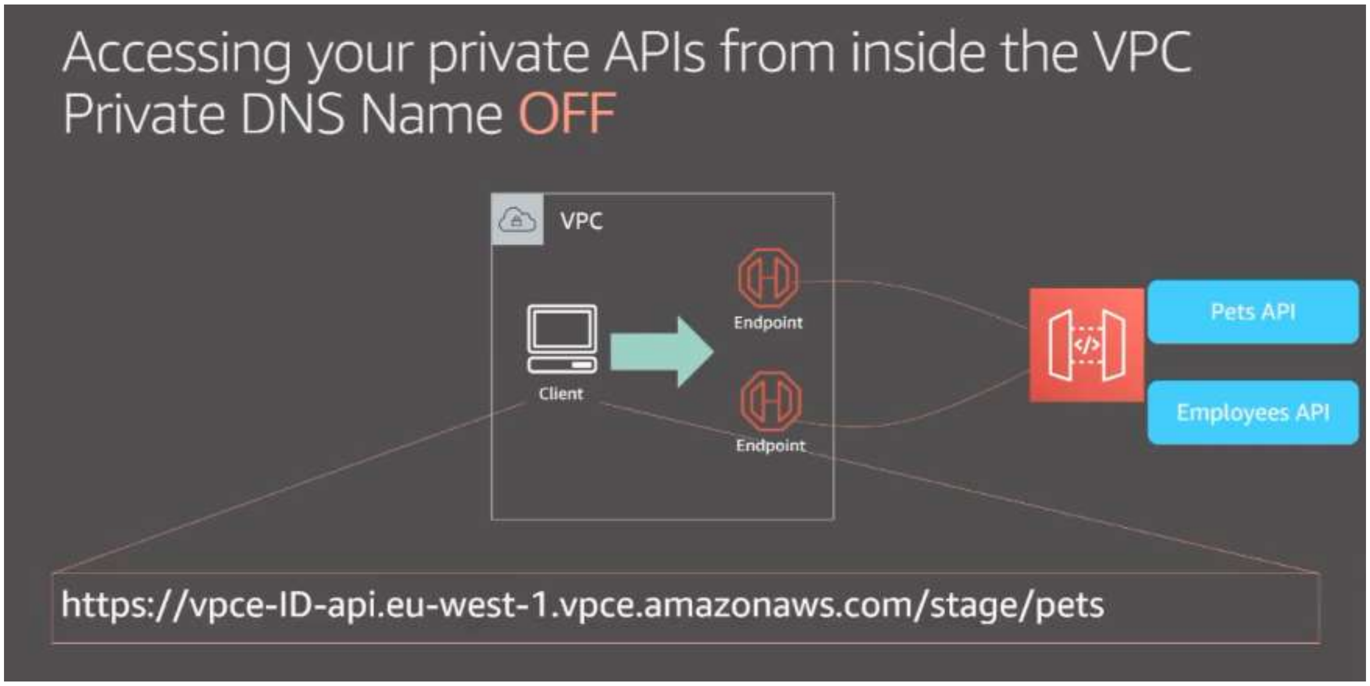
## Accessing your private APIs from inside the VPC
## Private DNS Name ON

https://petsAPIID.execute-api.eu-west-1.amazonaws.com/stage/pets
https://empAPIID.execute-api.eu-west-1.amazonaws.com/stage/employees

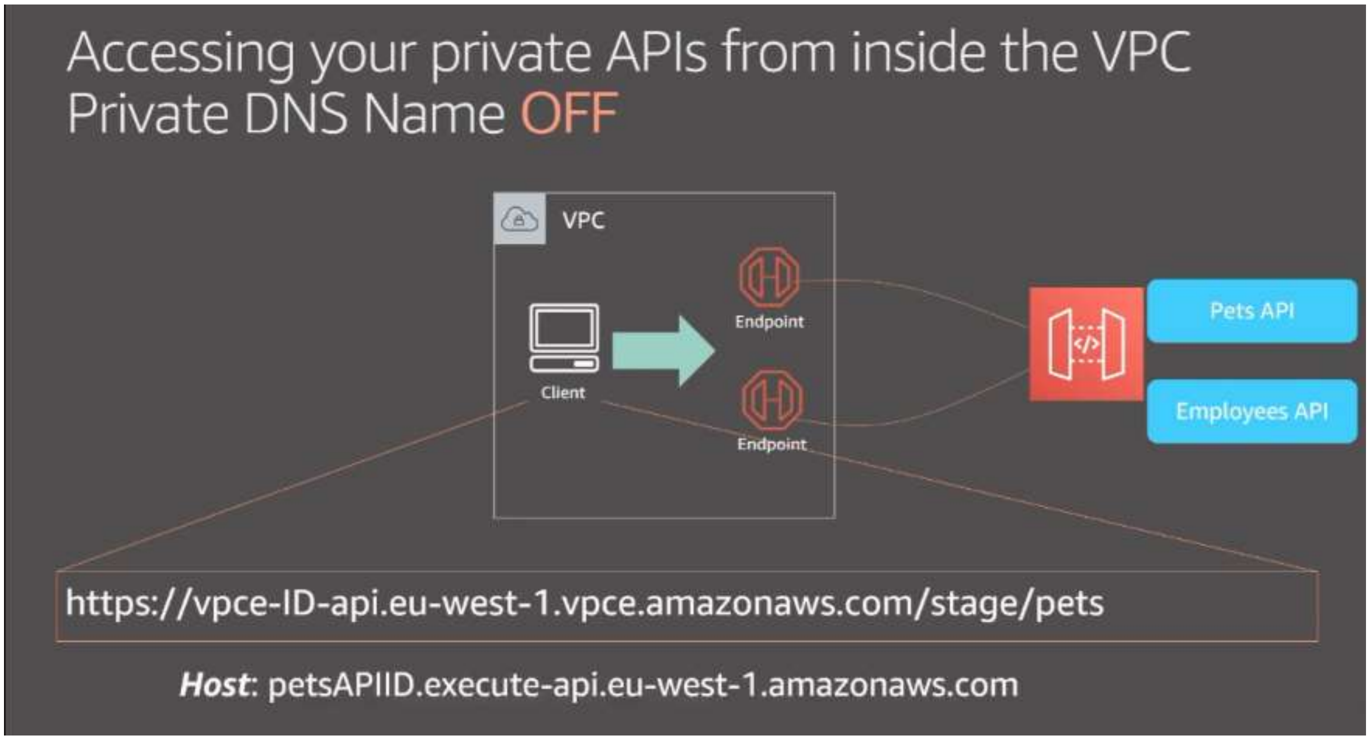


## Accessing your private APIs from inside the VPC
## Private DNS Name OFF

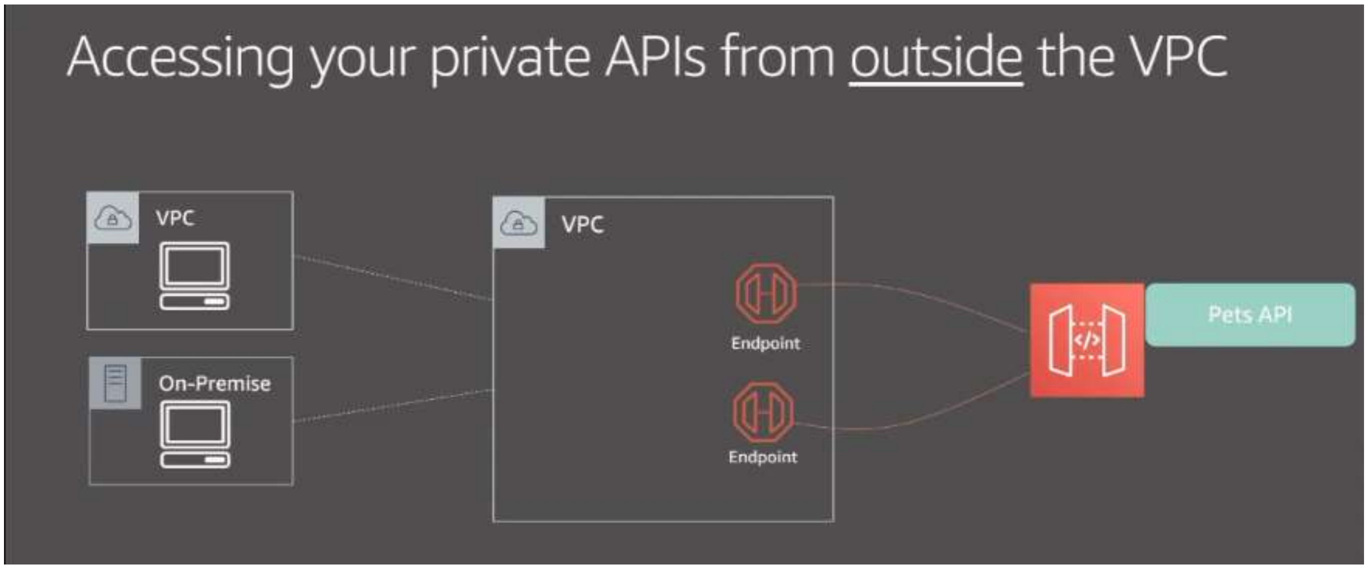
We can now no longer use the ***execute-api*** domain name anymore to connect to our VPC endpoints.


Accessing your private APIs from inside the VPC
Private DNS Name OFF
VPC
Client
Endpoint
Endpoint
Pets API
Employees API
https://vpce-ID-api.eu-west-1.vpce.amazonaws.com/stage/pets

We can instead use the publicly available VPC domain that ends in **.vpce** that is used for endpoints and it will use the VPC endpoint ID, the region, the domain, stage, method, route as above.

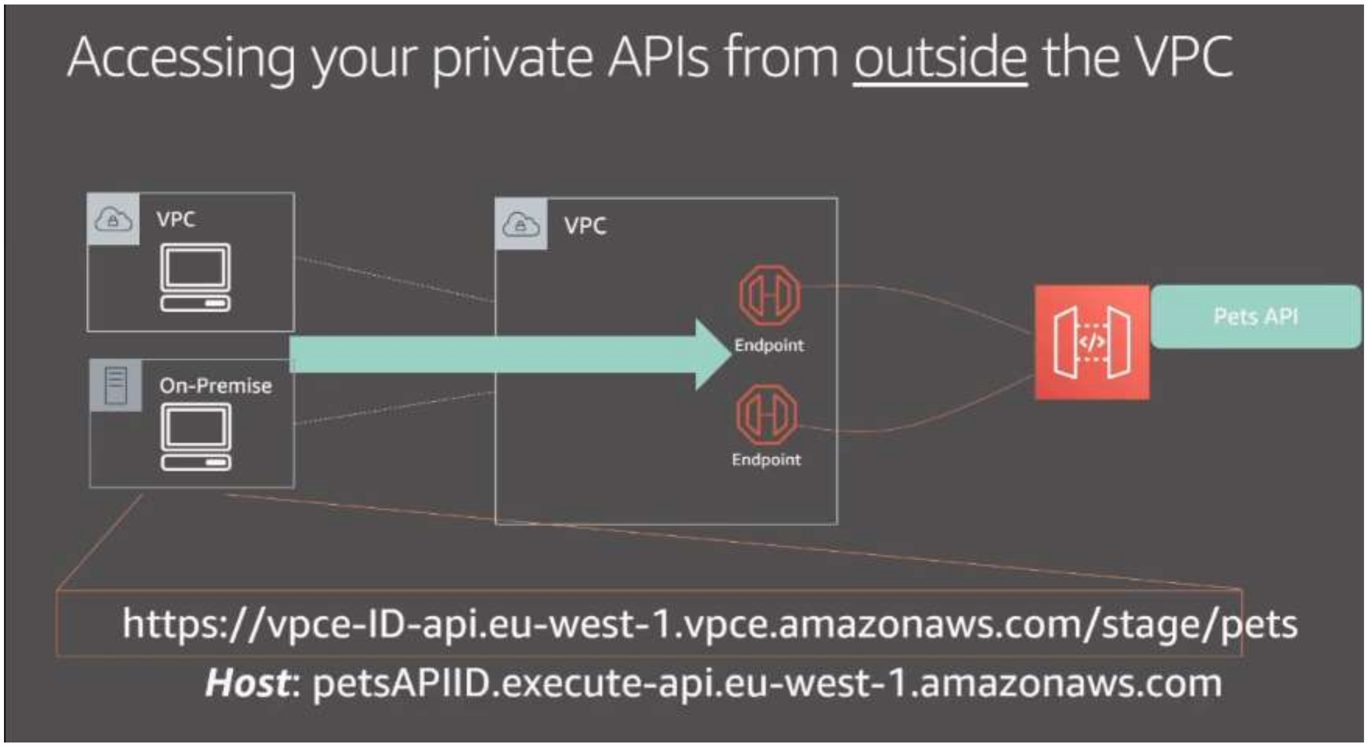

Accessing your private APIs from inside the VPC
Private DNS Name OFF
VPC
Client
Endpoint
Endpoint
Pets API
Employees API
https://vpce-ID-api.eu-west-1.vpce.amazonaws.com/stage/pets
Host: petsAPIID.execute-api.eu-west-1.amazonaws.com

We then add the Host header details so that the endpoints know which API project we are trying to hit.



## Accessing your private APIs from outside the VPC

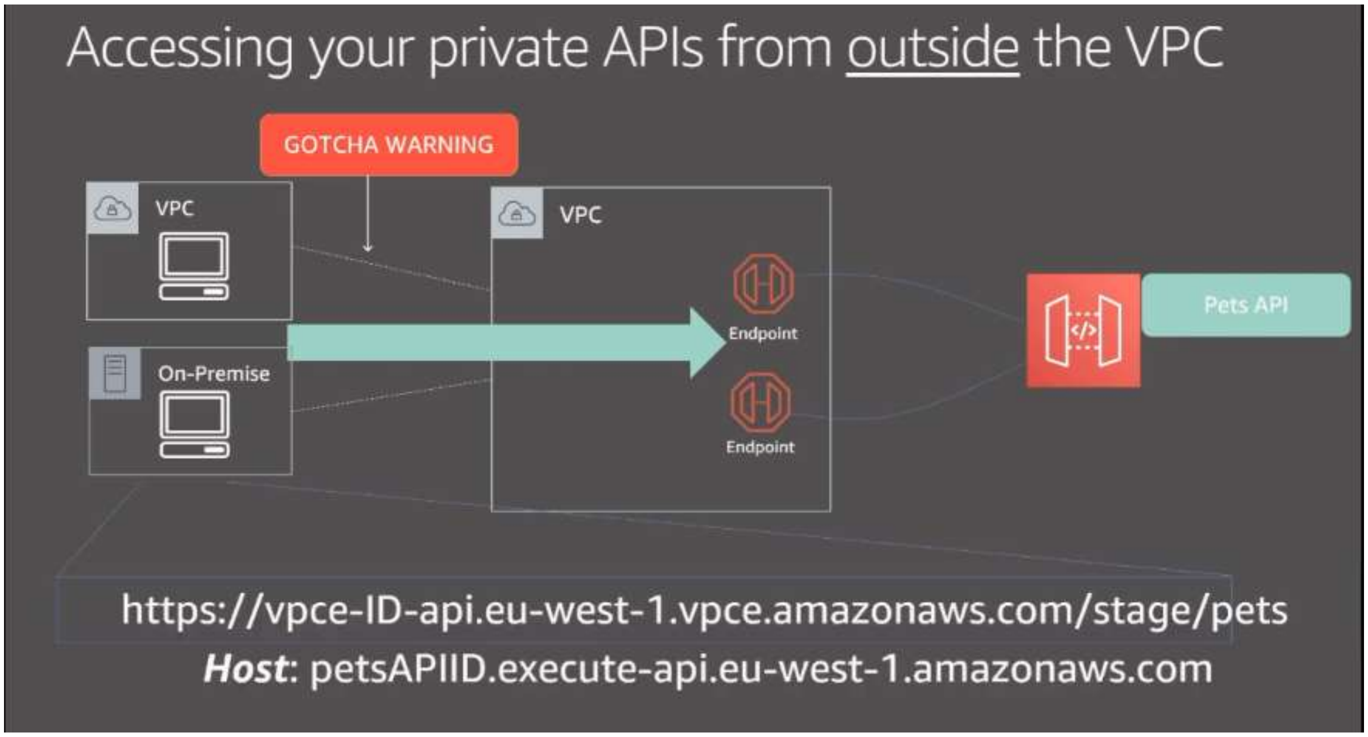
We don't want to use the internet for access



## Accessing your private APIs from outside the VPC

https://vpce-ID-api.eu-west-1.vpce.amazonaws.com/stage/pets

***Host***: petsAPIID.execute-api.eu-west-1.amazonaws.com



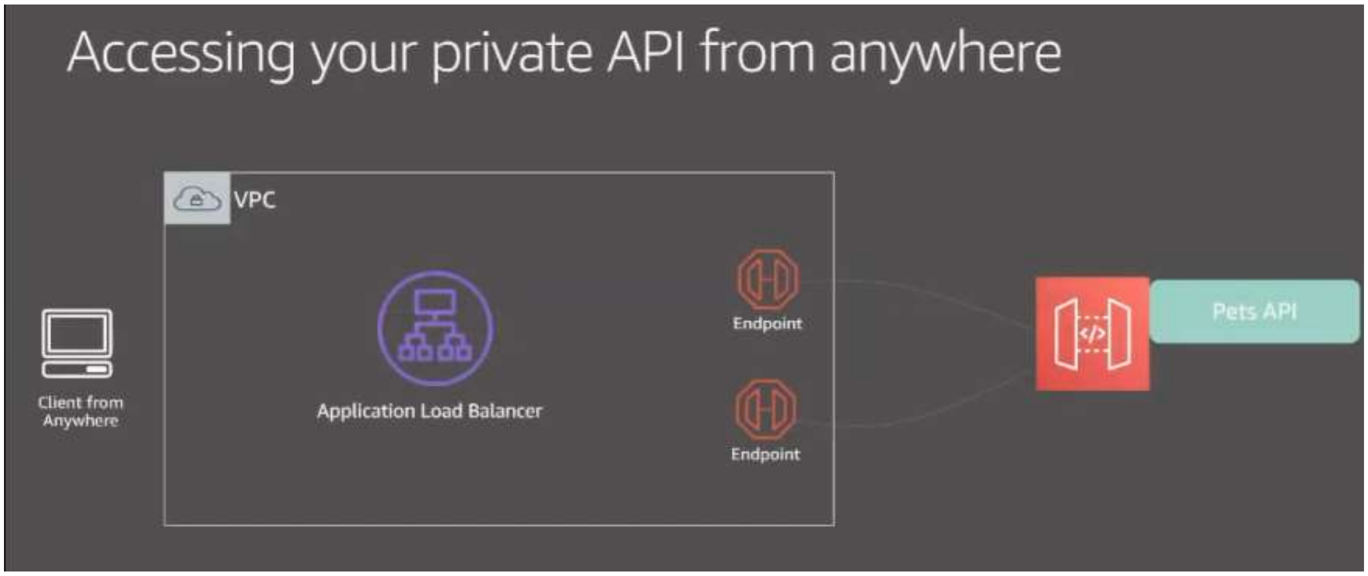
## Accessing your private APIs from outside the VPC

https://vpce-ID-api.eu-west-1.vpce.amazonaws.com/stage/pets

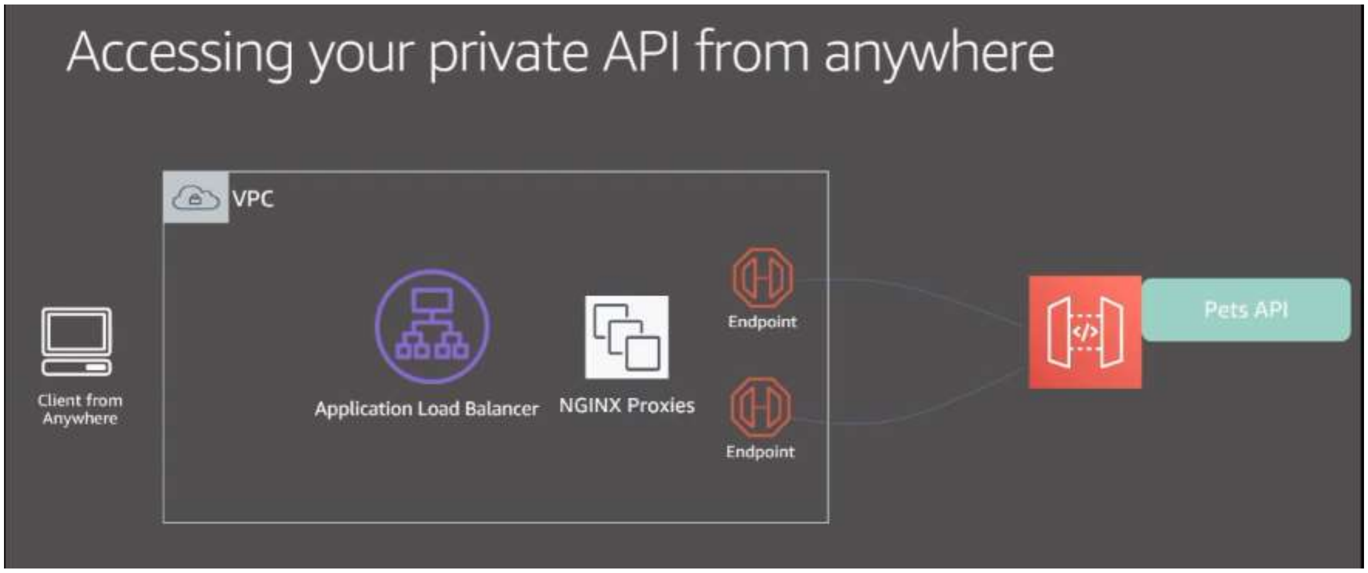
***Host***: petsAPIID.execute-api.eu-west-1.amazonaws.com

What happens if the VPC is in the same region? It will not hit those endpoints because it is not supported at the moment. The VPC has to be in another region and as to be hitting our VPC in a cross-region VPC peering manner

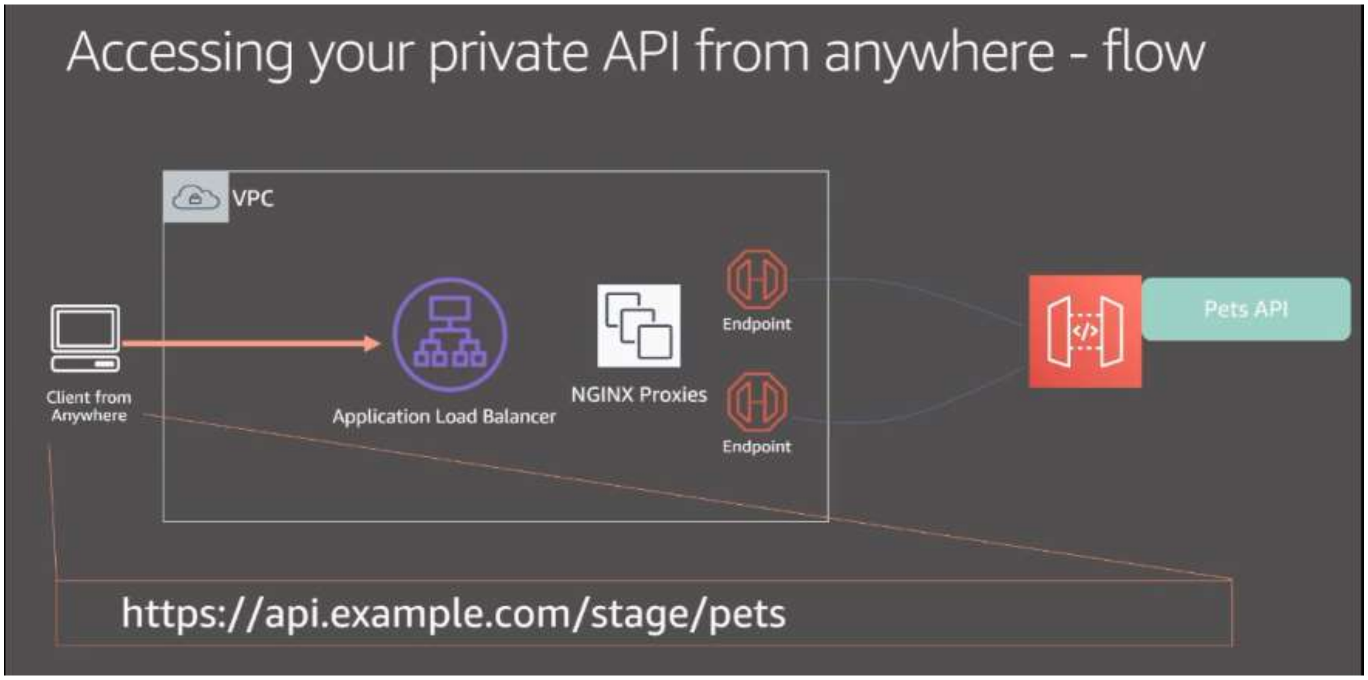
## What can customers do to solve this?



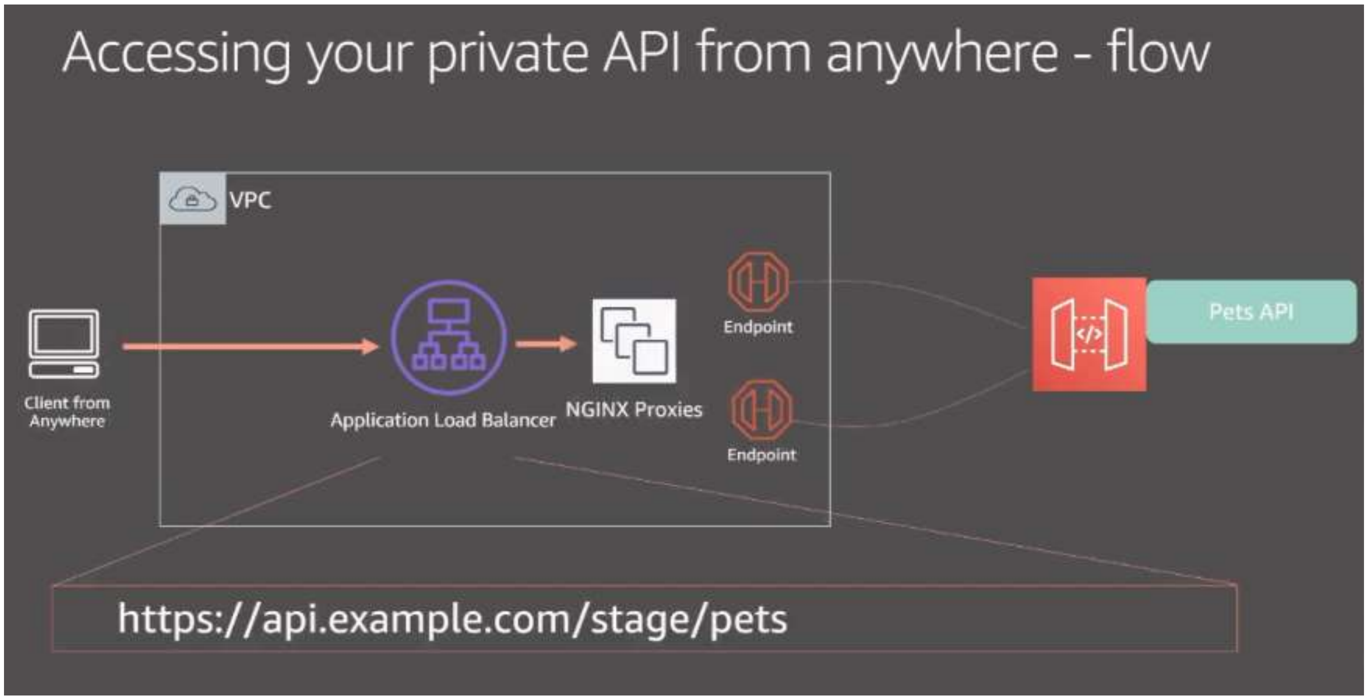
## Accessing your private API from anywhere


Accessing your private API from anywhere
VPC
Client from Anywhere
Application Load Balancer
NGINX Proxies
Endpoint
Endpoint
Pets API

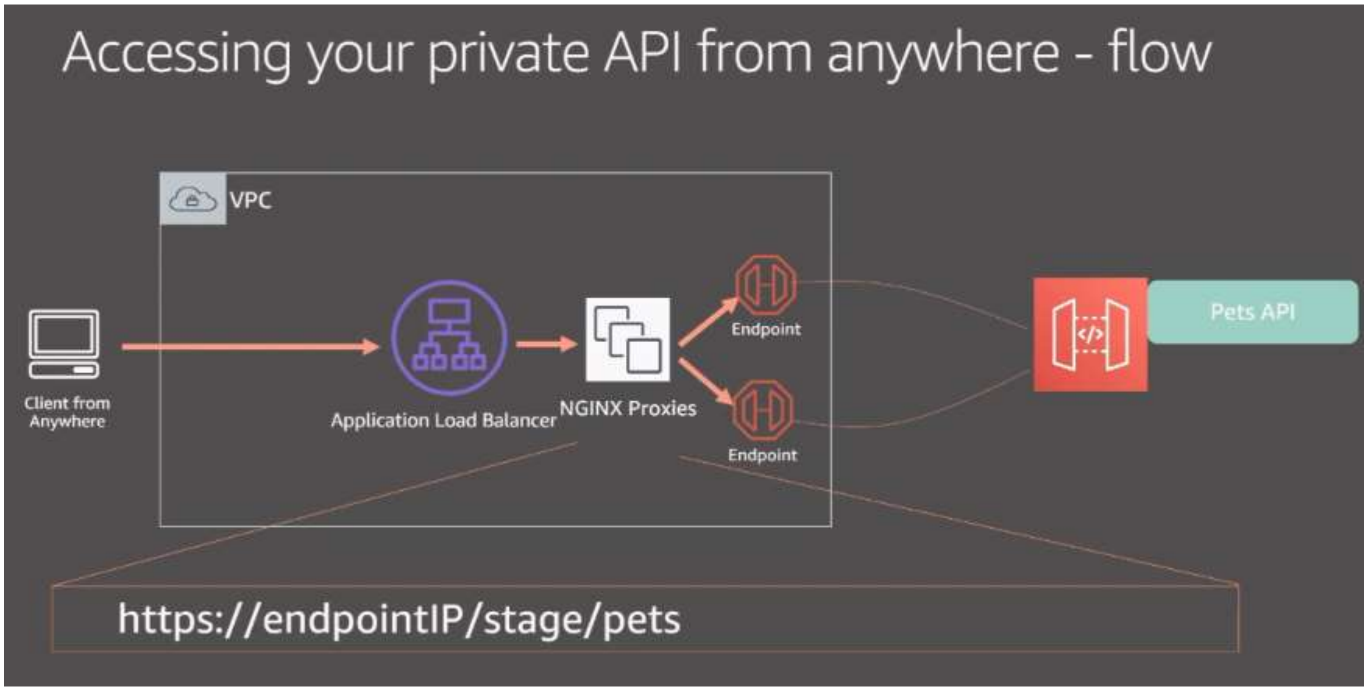
The ALB is fronting a lot of proxies servers to hit the endpoints


Accessing your private API from anywhere - flow
VPC
Client from Anywhere
Application Load Balancer
NGINX Proxies
Endpoint
Endpoint
Pets API
https://api.example.com/stage/pets

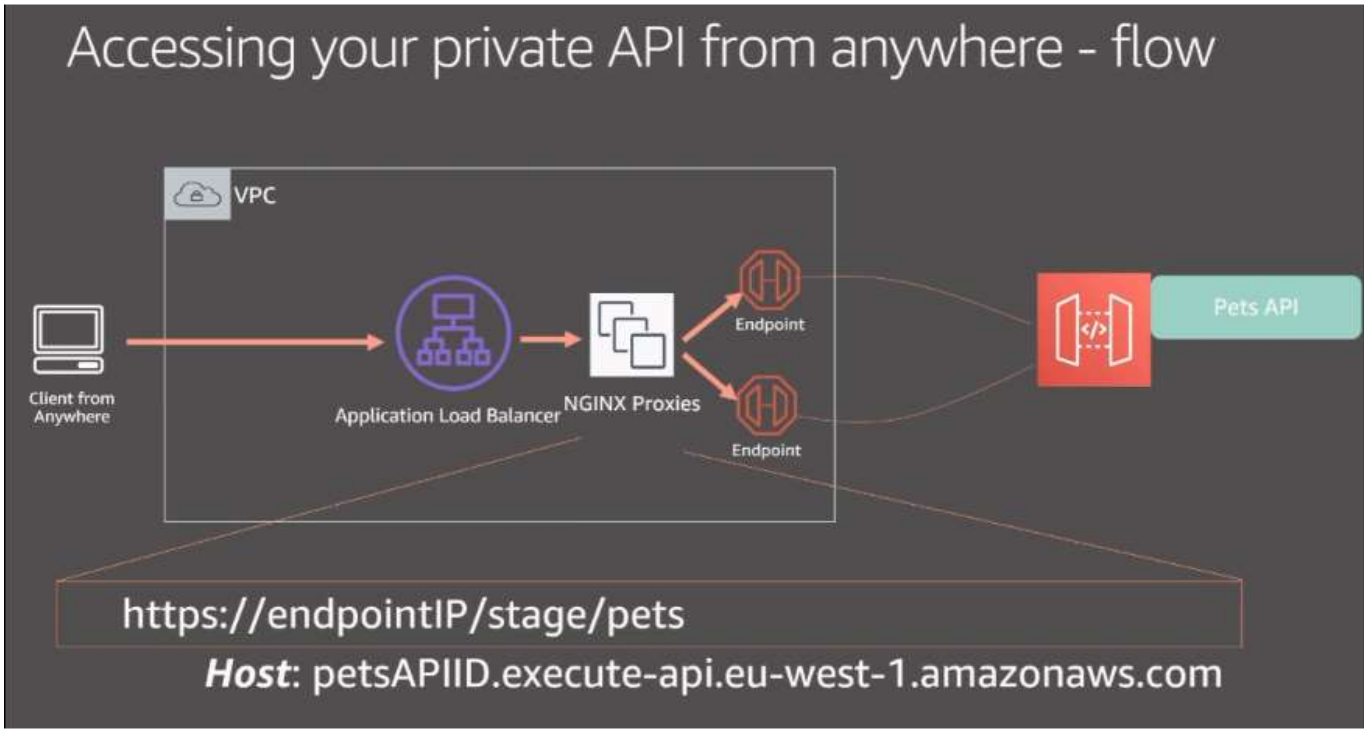
We can now use custom domains here since we are using an ALB


Accessing your private API from anywhere - flow
VPC
Client from Anywhere
Application Load Balancer
NGINX Proxies
Endpoint
Endpoint
Pets API
https://api.example.com/stage/pets

The ALB doesn't do anything and just forwards the request to the Nginx proxy


Accessing your private API from anywhere - flow
VPC
Client from Anywhere
Application Load Balancer
NGINX Proxies
Endpoint
Endpoint
Pets API
https://endpointIP/stage/pets

The proxy now uses rules (created off of the path **/pets** or the header information) to add the host header with the specific API Gateway project URL as below



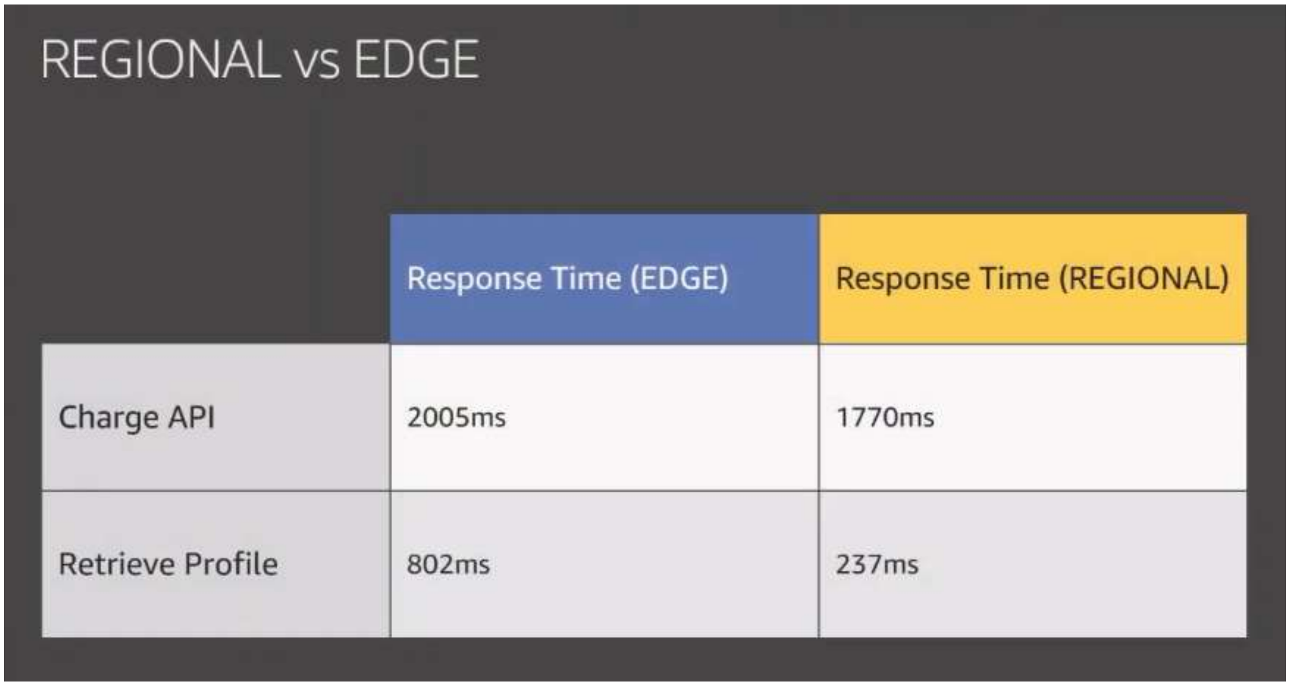
## Accessing your private API from anywhere - flow

https://endpointIP/stage/pets

***Host***: petsAPIID.execute-api.eu-west-1.amazonaws.com

The VPC endpoints now know how to get to the API gateway project and forward traffic to it. We have now moved the complexity away from the client to the centralized network environment as suggested by the best practises.

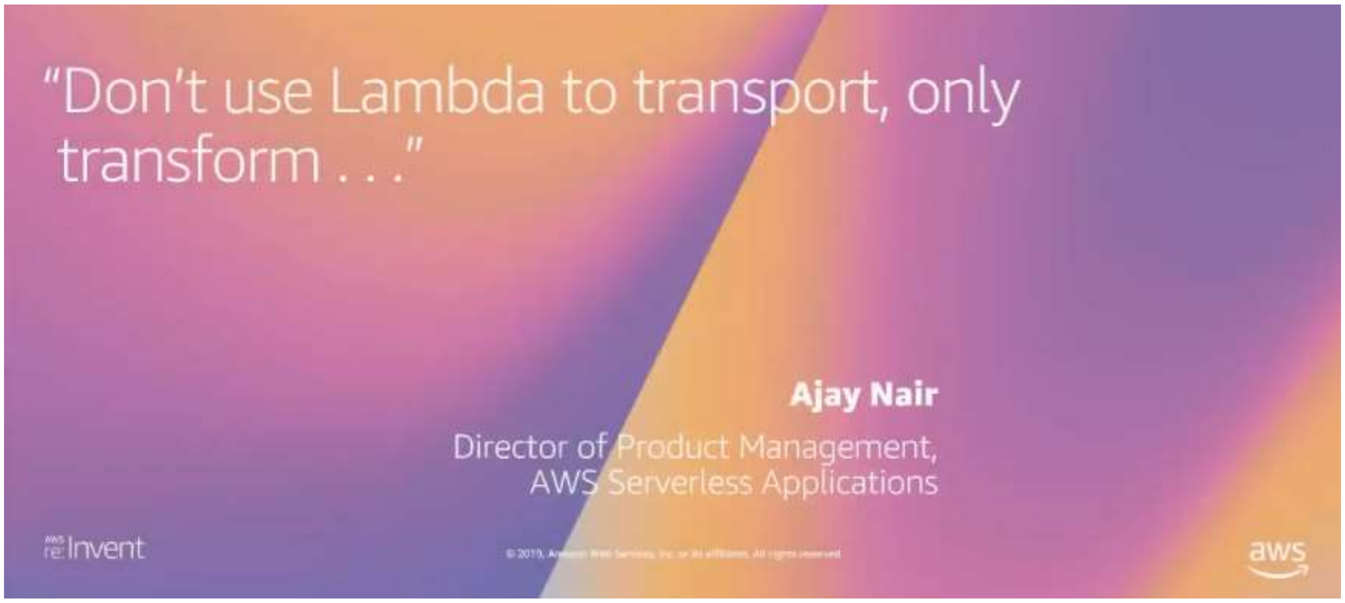
## Public API Gateway best practices



### REGIONAL vs EDGE

| | Response Time (EDGE) | Response Time (REGIONAL) |
|---|---|---|
| Charge API | 2005ms | 1770ms |
| Retrieve Profile | 802ms | 237ms |

There are 2 different kinds of Public API Gateway endpoints, **regional** and **edge-optimized**, they have very different response times. The **Edge-optimized** API Gateway will create an AWS CloudFront distribution for you, this will add to your response time. If you know the location of your clients, then choose the **regional** API Gateway option. But serverless applications will default to edge-optimized, you will need to change that yourself.



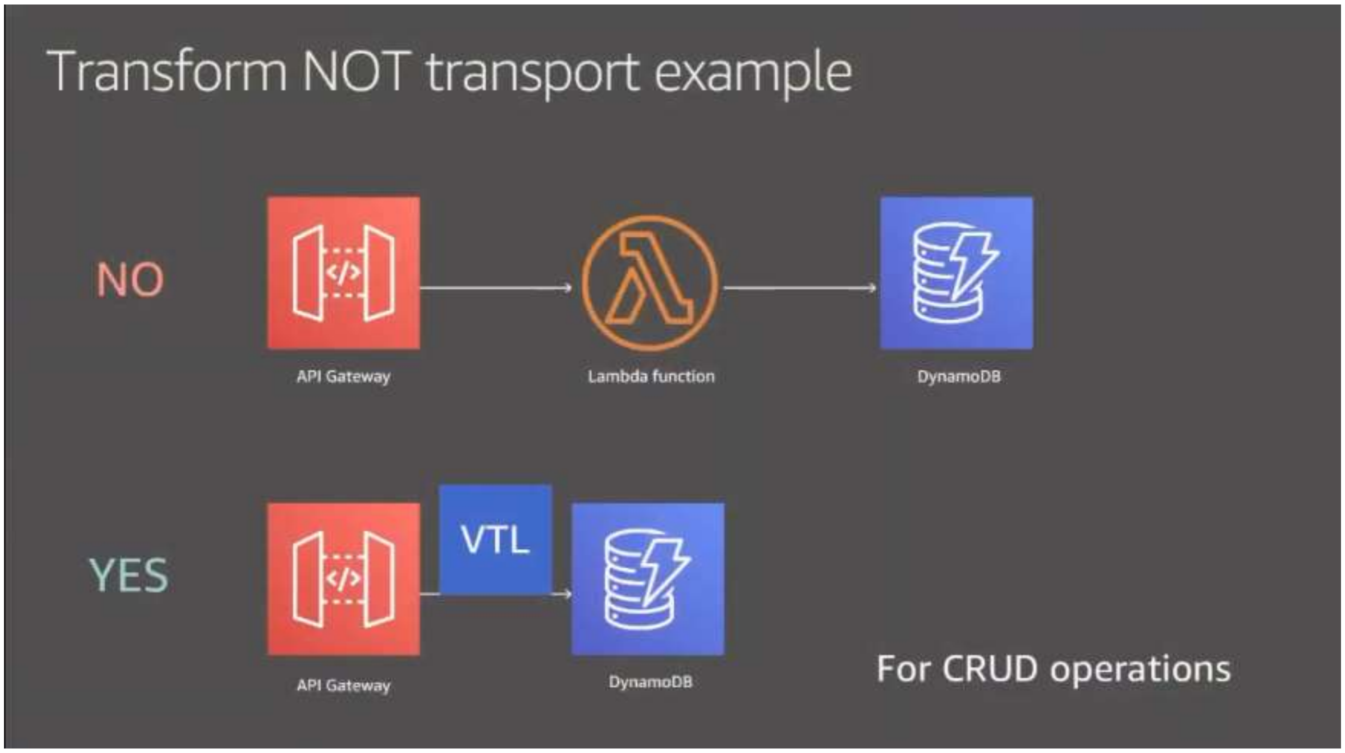
"Don't use Lambda to transport, only transform . . ."

Ajay Nair

Director of Product Management,
AWS Serverless Applications

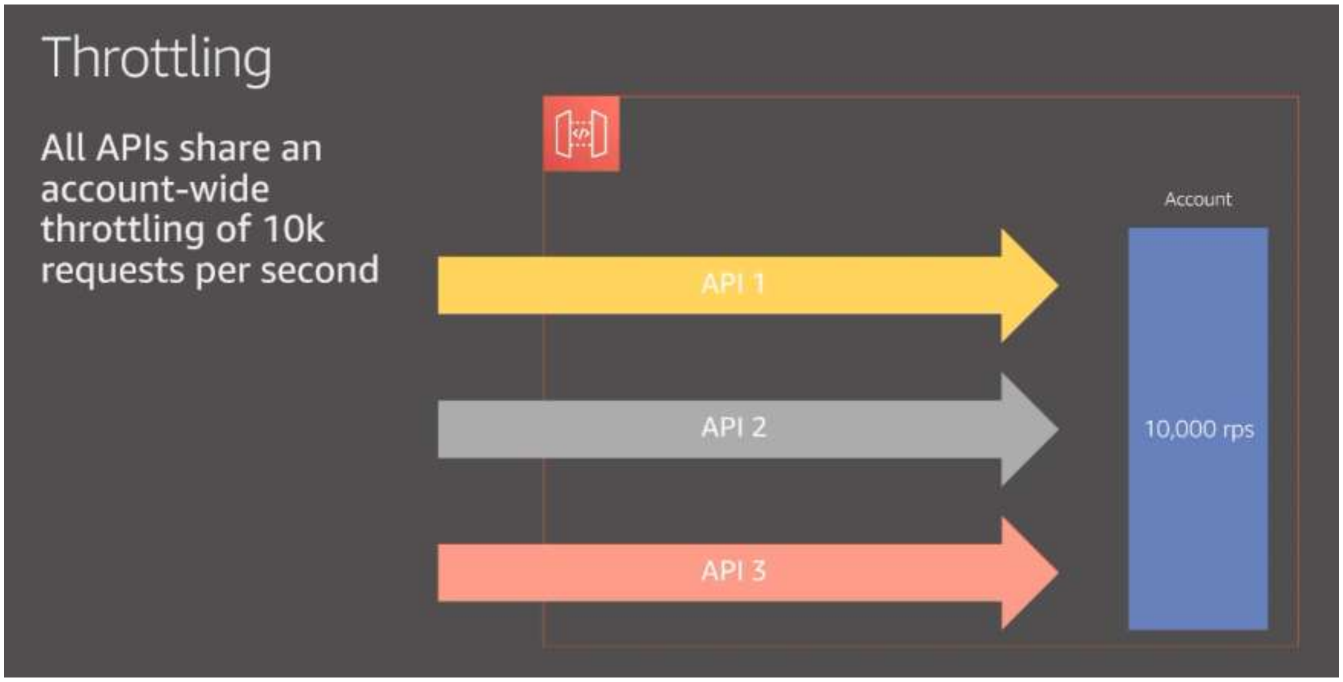
re:Invent
aws

There are cases where you actually don't need to use a lambda



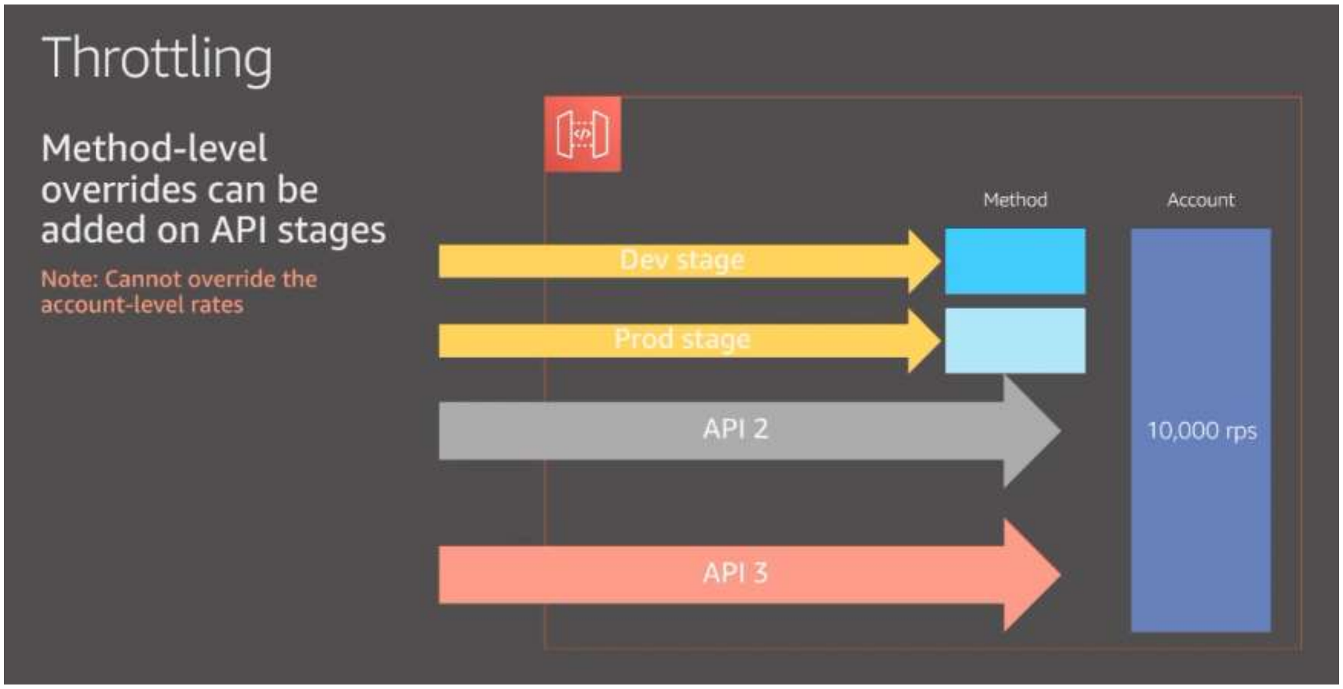
## Transform NOT transport example

You can instead set up your API Gateway to directly access DynamoDB itself without going through a Lambda function. You set up the API Gateway route as an AWS Proxy service and use a feature called VTL (velocity templating language used as a mapping function) to dump the object directly in DynamoDB.


## Throttling

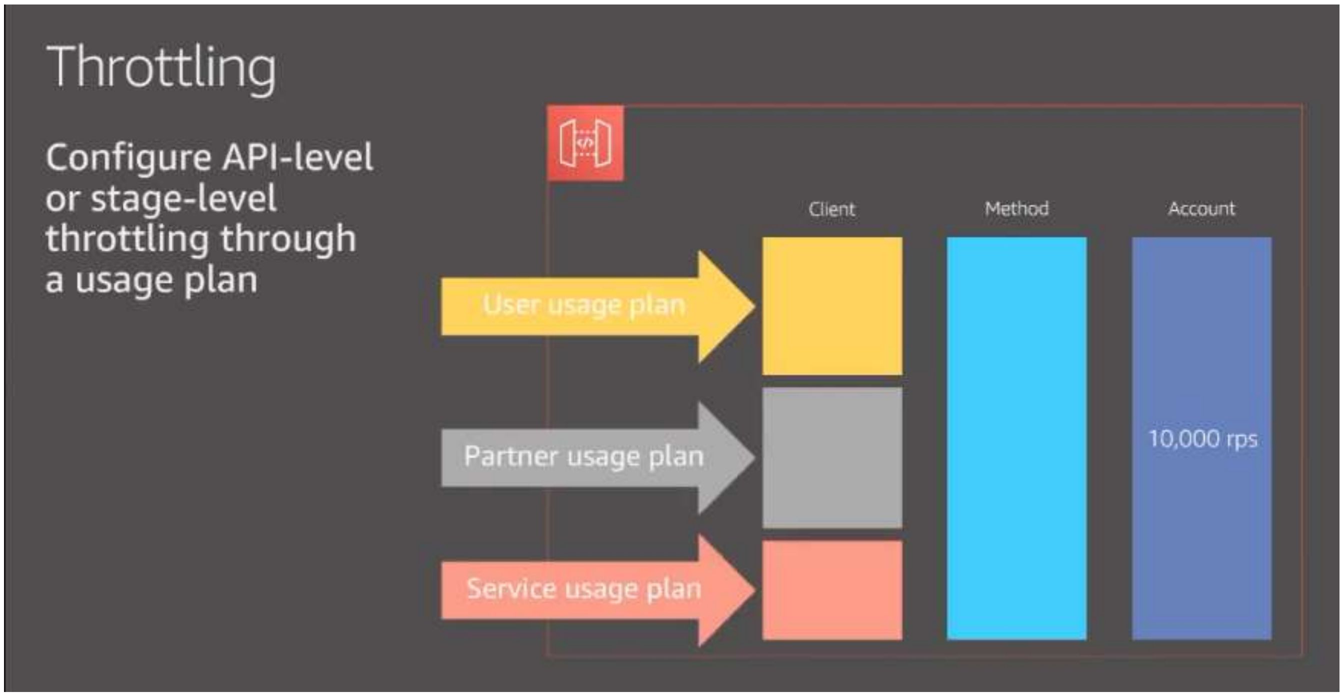
All APIs share an account-wide throttling of 10k requests per second

Because of the 10,000 RPS account-wide APIs access limit (and a 5000 RPS burst limit), you don't want your most popular API Gateway route to suck up all the RPS and not allow the other routes to function. We need to use throttling to prevent this from happening.

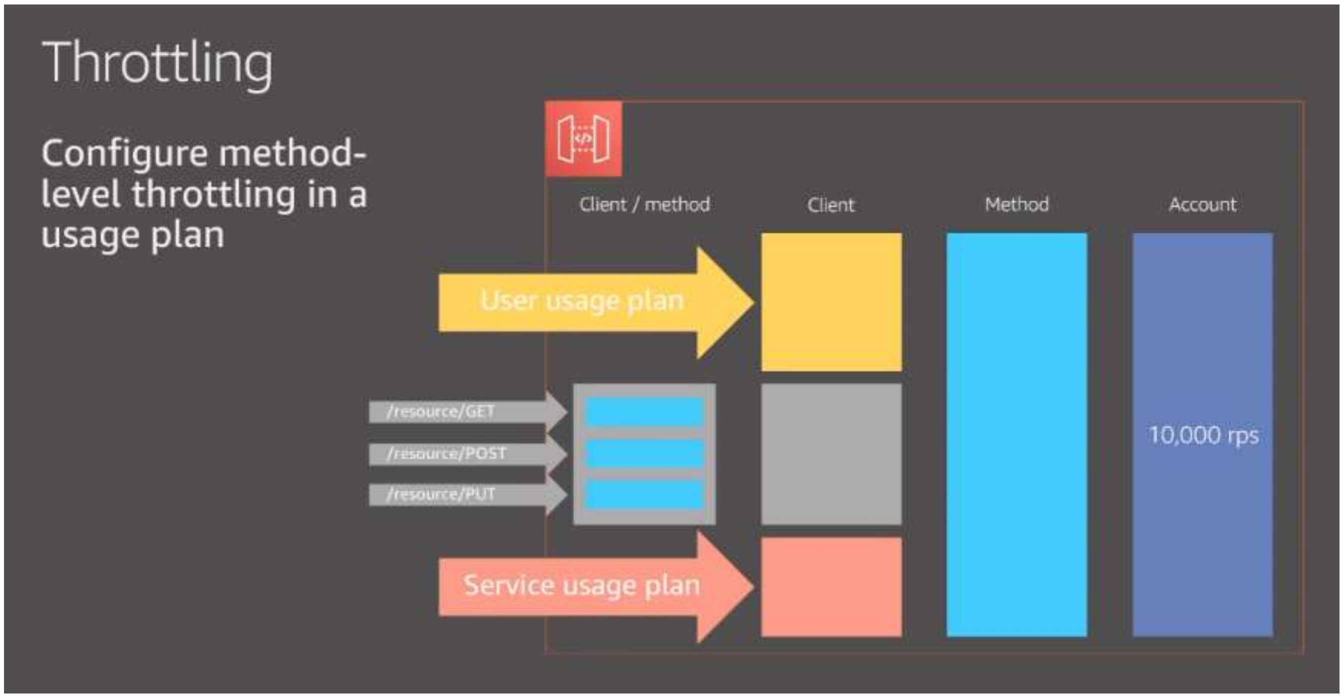

## Throttling

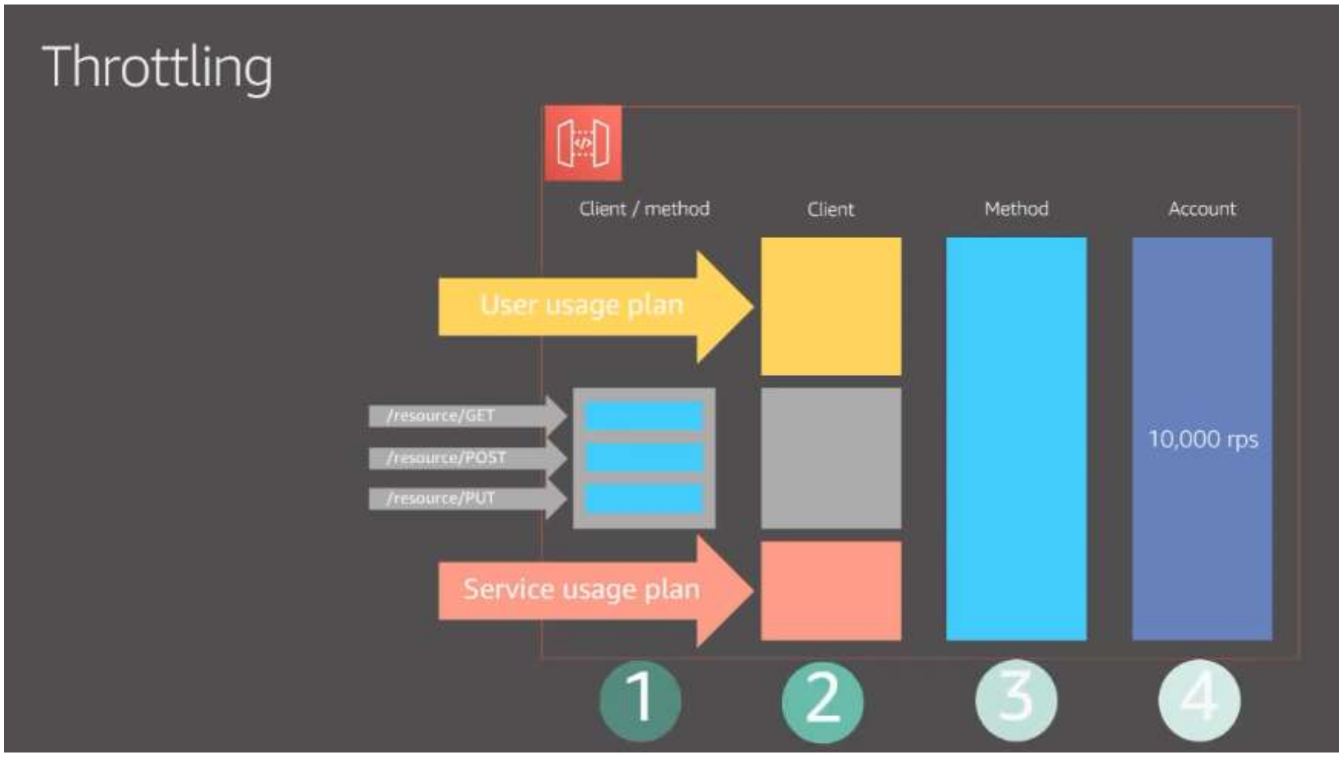
Method-level overrides can be added on API stages

Note: Cannot override the account-level rates

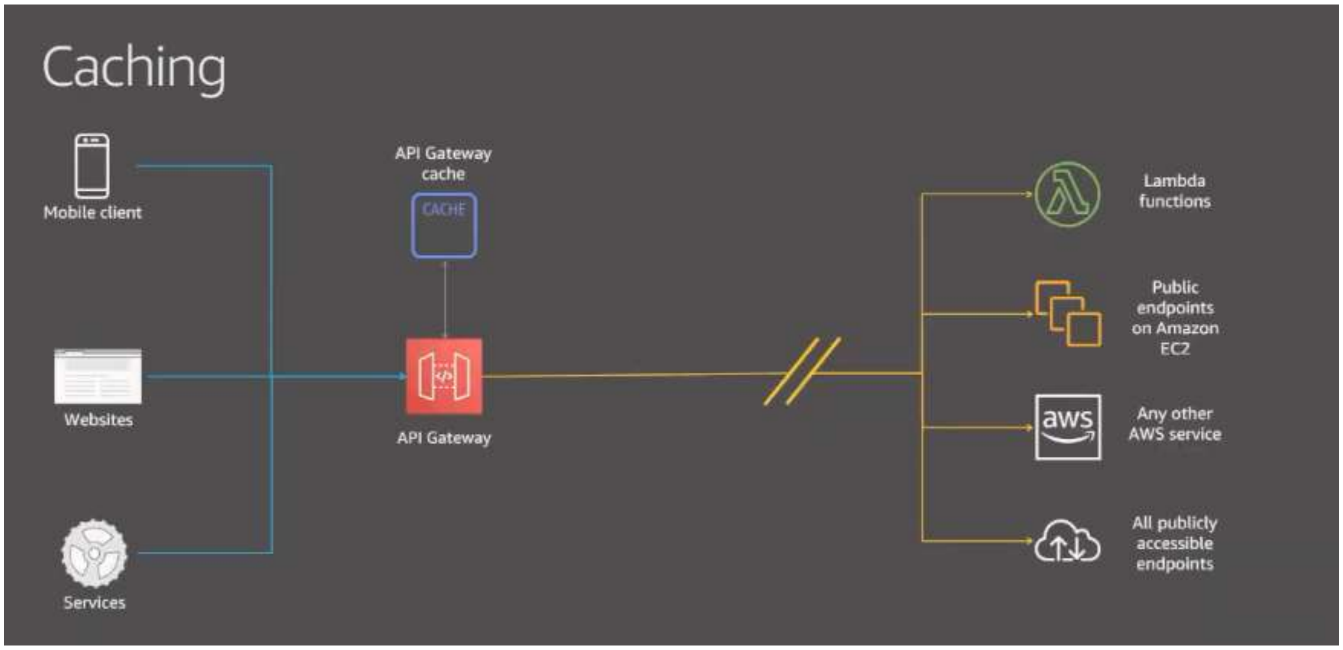


You can instead create usage plans for your different APIs and throttle the different levels like 1500 RPS for some and 5000 RPS for other usage plans.



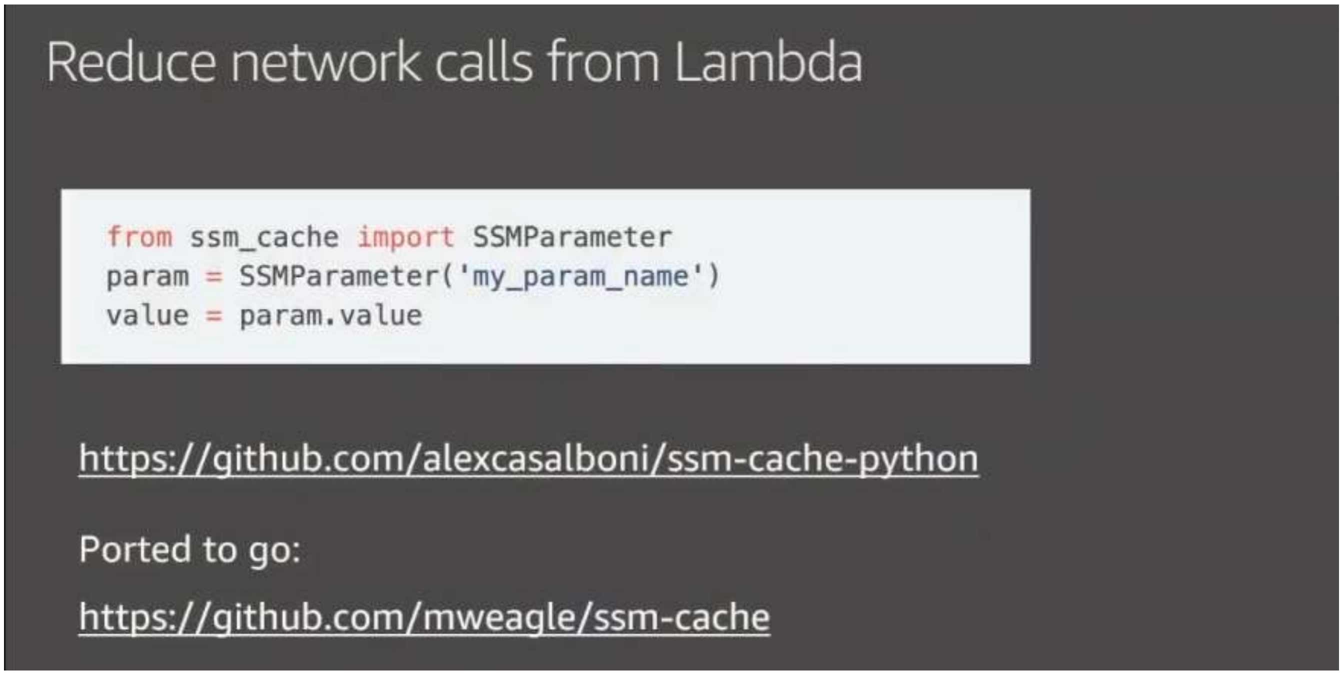


Throttling settings are applied in a specific order shown above



We can use API Gateway caching to cache path, headers, query strings etc to increase the speed and load on our backend services.

## Miscellaneous


## Reduce network calls from Lambda

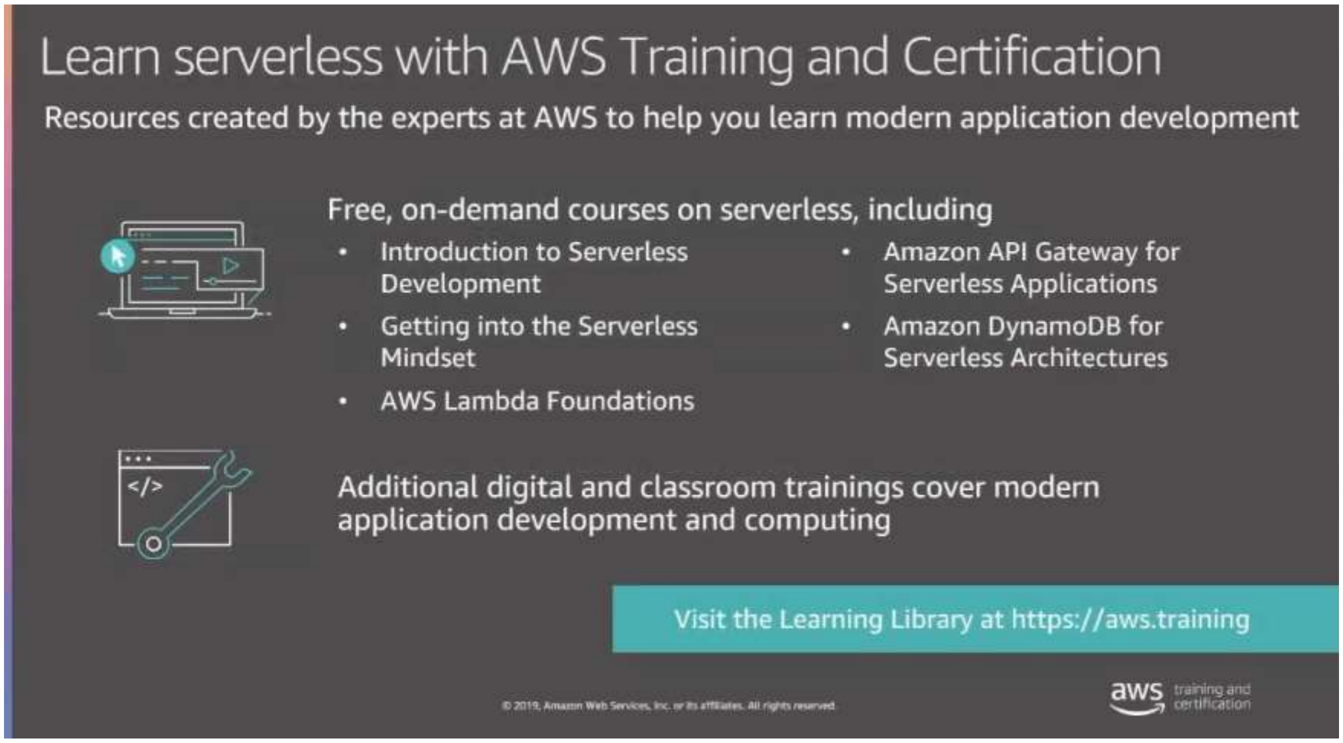
```python
from ssm_cache import SSMParameter
param = SSMParameter('my_param_name')
value = param.value
```

https://github.com/alexcasalboni/ssm-cache-python

Ported to go:

https://github.com/mweagle/ssm-cache

This is a caching service to cache parameter store and secrets values to decrease your lambda calls, it is available in Python and Go.


## Learn serverless with AWS Training and Certification

Resources created by the experts at AWS to help you learn modern application development

Free, on-demand courses on serverless, including

- Introduction to Serverless Development
- Getting into the Serverless Mindset
- AWS Lambda Foundations
- Amazon API Gateway for Serverless Applications
- Amazon DynamoDB for Serverless Architectures

Additional digital and classroom trainings cover modern application development and computing

Visit the Learning Library at https://aws.training

aws training and certification