Creating a bridge between two accounts on AWS

Imagine that you have two accounts on AWS: PRODUCTION; where you will install all final and stable version of the applications of your business and INTERNAL; where you put all the development version of your products for testing purposes. You can have a lot of accounts for validating the development, making end-to-end (e2e) tests and others for other purposes.

Now, imagine that you have an application inside a ec2 instance and this will collect information about your PRODUCTION applications. It can be a custom centralized monitoring stack to monitor the health of your systems and other metrics or it can be just another application that access sensitive data for internal uses.

The big problem here is that you have to setup an application load balancer (ALB) to access these data. If you build an network load balancer (NLB), it will be only available internally on the PRODUCTION account and the development team or the stakeholders will not have access to it. With the ALB it will be publicly accessible over the internet. Nevertheless you whitelist it, it's risky.

The solution for this is to setup a bridge between two AWS accounts. This article shows you how. Linked with this article there is a Terraform code on github that you can put information of your solution and build this bridge. But first let's see how this works.

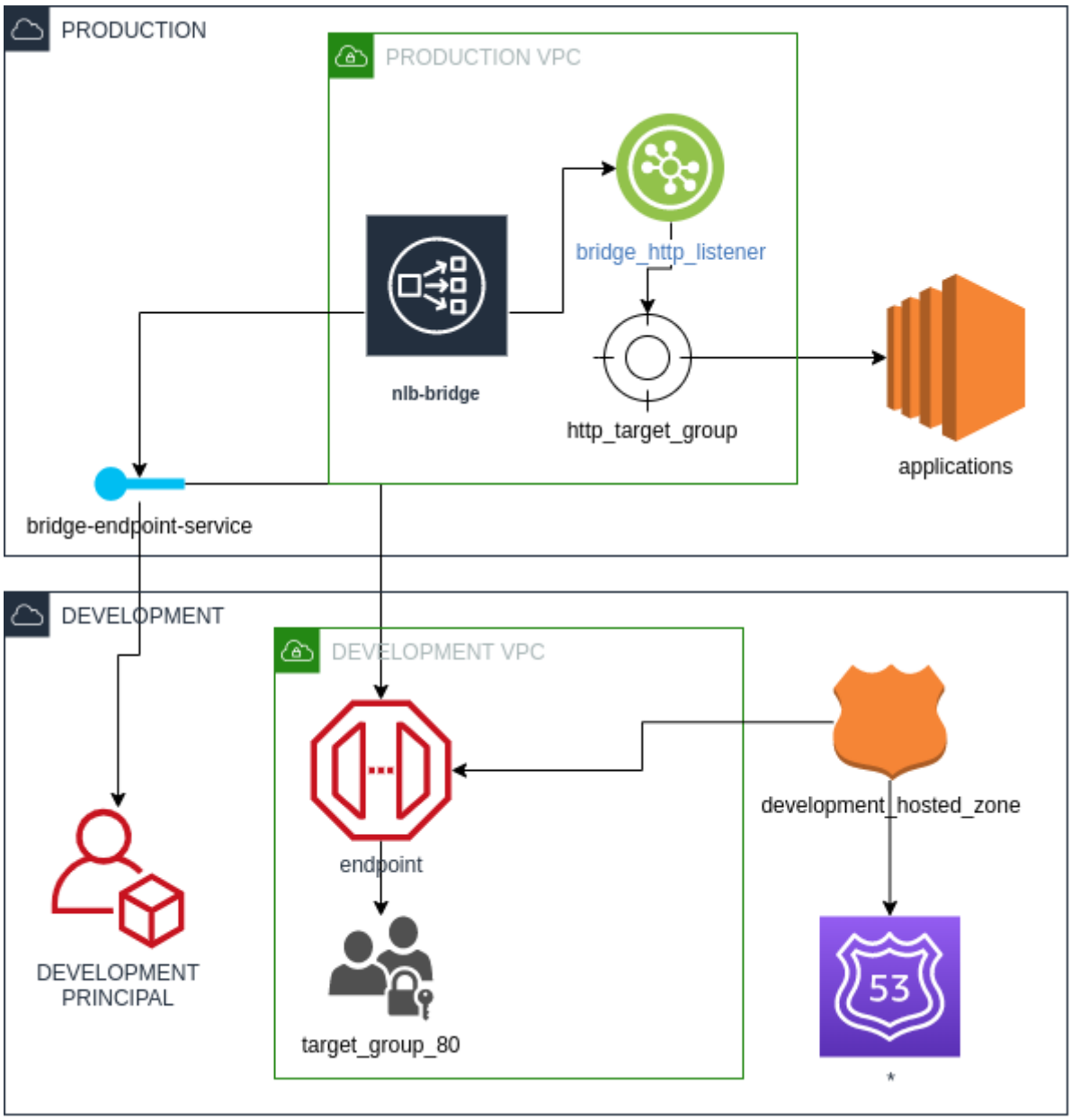
Assumptions

There are some things that you have to keep in mind before building this solution:

1. You have an ec2 instance on PRODUCTION account
2. You have VPCs on each account that have the same number of subnets
3. You have a Hosted Zone configured for the INTERNAL account
4. You have a profile for PRODUCTION and INTERNAL accounts
5. You have Terraform installed on version 0.15 (at the moment I'm writing this there's the 1.0 version)

Building

The diagram below shows the solution:



In the top of the diagram is located the ec2 instances with the applications with the monitoring stack or the other solution designed for retrieving information from the business applications. The way that other applications send information is up to the system you are developing. The boundary is the VPC Endpoint Service that creates a service for the NLB.

In the DEVELOPMENT part, there is a VPC Endpoint that will create an endpoint for the service on PRODUCTION account and for this to be possible we need, to add the DEVELOPMENT principal to the VPC Endpoint Service. For the endpoint be able to be accessible using a browser, it's needed a Target Group for the 80 port. The Hosted Zone and the record are there to make a more friendly DNS name than the VPC Endpoint will provide.

Now that's the moment some developers says: What? Don't worry, I will explain the concepts behind each component here and show some Terraform code for this. Just relax, it won't be hard.

PRODUCTION

In the PRODUCTION account will be needed the following:

- **REQUIRED**
  - ec2-instance
  - vpc
  - subnets

    - DEVELOPMENT account Principal ARN
- **BUILD**:
    - NLB
    - NLB Listener
    - NLB Target Group
    - VPC Endpoint Service

NLB

A Load Balancer is ... . There are x types of it: Classic Load Balancer, Application Load Balancer, and Network Load Balancer. The Classic Load Balancer (LB) is used to ... . The Application Load Balancer (ALB) does ... . I am using a Network Load Balancer (NLB) because ... . The code below shows how to build the NLB to the solution:

```
// Network Load Balancer
resource "aws_lb" "bridge" {
  provider           = aws.production_account
  name               = "${var.project_name}-bridge"
  internal           = true
  load_balancer_type = "network"

  subnets      = var.source_subnet_ids
  idle_timeout = 60
}
```

There are another two resources to create:

1. The Listener that will ...
2. The Target Group that will ...

Here is the code for those resources:

```
// HTTP Listener
resource "aws_lb_listener" "bridge_http_listener" {
  provider          = aws.production_account
  load_balancer_arn = aws_lb.bridge.arn
  port              = 80
  protocol          = "TCP"

  default_action {
    type             = "forward"
    target_group_arn = aws_alb_target_group.bridge_http_target_group.arn
  }
}

// HTTP Listener Target Group
resource "aws_alb_target_group" "bridge_http_target_group" {
  provider = aws.production_account
  name     = "${var.project_name}-bridge-http-tg"
  protocol = "TCP"
  port     = 80
  vpc_id   = var.source_vpc_id
}

// HTTP Listener Target Group Attachment
resource "aws_alb_target_group_attachment" "bridge_http_tg_attachment" {
```

```
  provider         = aws.production_account
  target_group_arn = aws_alb_target_group.bridge_http_target_group.arn
  target_id        = var.source_instance_id
}
```

VPC Endpoint Service

The VPC Endpoint service is a simple resource that ... . For this solution, it was used the ... .

Here's the code for that:

```
// Endpoint Service
resource "aws_vpc_endpoint_service" "bridge_service" {
  provider = aws.production_account

  acceptance_required = false
  network_load_balancer_arns = [
    aws_lb.bridge.arn
  ]

  tags = {
    Name = "${var.project_name}-bridge-service"
  }
}

// Endpoint Service Principal
resource "aws_vpc_endpoint_service_allowed_principal" "bridge_service_principal"
{
  provider = aws.production_account

  principal_arn           = var.service_allowed_principal
  vpc_endpoint_service_id = aws_vpc_endpoint_service.bridge_service.id
}
```

DEVELOPMENT

In the DEVELOPMENT account will be needed the following:

- **REQUIRED**
  - vpc
  - subnets
  - Hosted Zone
- **BUILD**:
  - VPC Endpoint
  - Target Group
  - Route 53 Record

VPC Enpoint

Differently from the VPC Endpoint Service, the VPC Endpoint creates ... The following code shows the creating of the VPC Endpoint:

```
// Endpoint
resource "aws_vpc_endpoint" "bridge_endpoint" {
  provider = aws.internal_account
```

```
  vpc_id     = var.target_vpc_id
  subnet_ids = var.target_subnet_ids
  security_group_ids = [
    aws_security_group.bridge_endpoint_security_group.id
  ]

  service_name      = aws_vpc_endpoint_service.bridge_service.service_name
  auto_accept       = true
  vpc_endpoint_type = "Interface"

  tags = {
    Name = "${var.project_name}-bridge-endpoint"
  }

  depends_on = [
    aws_vpc_endpoint_service.bridge_service
  ]
}
```

Also, our VPC Endpoint needs a Target Group to allow traffic from the port 80. This Target Group is different from the PRODUCTION because ... . Here's the code of the Target Group:

```
// Security Groups to only allow HTTP and SSH Traffic
resource "aws_security_group" "bridge_endpoint_security_group" {
  provider = aws.internal_account

  name   = "${var.project_name}-bridge-endpoint-sg"
  vpc_id = var.target_vpc_id

  ingress {
    description = "HTTP Connection"
    protocol    = "tcp"
    from_port   = 80
    to_port     = 80
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "${var.project_name}-bridge-endpoint-sg"
  }
}
```

Route 53 Record

The last resource, but not the least, is the Route 53 record. It will need a pre-configured Hosted Zone on the DEVELOPMENT account to be built. It will build ... . The code bellow shows how to configure the record:

```
// Route 53 Record
resource "aws_route53_record" "application_record" {
  provider = aws.internal_account
  name     = "*"
  type     = "CNAME"
  zone_id  = var.target_hosted_zone_id
  records = [
    aws_vpc_endpoint.bridge_endpoint.dns_entry[0].dns_name
  ]
  ttl = 300
}
```

Note that the name is just "*****". I recommend for you to have a web server configured (e.g. Apache, ngnix) on the ec2 machine and add domains to the deployed applications. It will be possible to access your deployed applications with the predetermined prefix and the suffix defined on the Hosted Zone. Example:

Hosted Zone Prefix:

- application.internal.hub.com

Applications on the ec2 machine:

- logs
- metrics
- manager

Outputs:

- logs.application.internal.hub.com
- metrics.application.internal.hub.com
- manager.application.internal.hub.com

The configurations of the domains are not on the github repository, because they are not part of the solution, but part of the application that is being developed.

Files

The project on github is written in Terraform. Before running anything, configure the following files:

- `applications.tfvars`: The input information
- `main.tf`: Providers information

After configuring, run:

- `terraform init` to download plugins and the provider information
- `terraform plan` to check the resources that will be created

Be aware that you must have awscli pre-configured on the machine.

At the end of the process, Terraform will show you some outputs that are on the `outputs.tf` file. The outputs are:

- The fully qualified domain name of the Route53 Record
- Examples of three example endpoints

Conclusion

Information security is important and there are several ways to implement. This is a solution that does not use a sophisticated mechanism or measures, it uses a safe place that already exists, the DEVELOPMENT account, that is accessed only by the development team and some stakeholders.