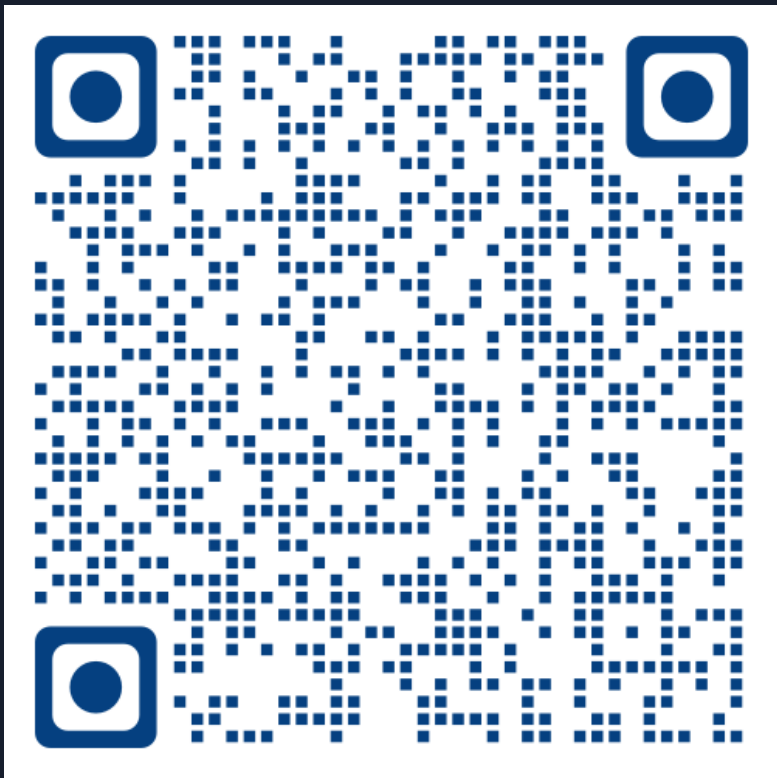


AWS Terraform Small Environment

→ Maxim Crucovschii, Cloud Automation Engineer



→ <https://github.com/mcrucovschii/SmallEnvironment>
mcrucovschii@gmail.com

Contents

[Statement Of Work](#)

[Architecture](#)

[Solution](#)

[Code](#)

[Questions](#)

Statement Of Work

→ Assignment

Small Environment, the Requirements:

2 LBs (Custom Hardened Linux AMI)

2 Application-Server (default AWS AMI)

3 DB-Nodes (default AWS AMI)

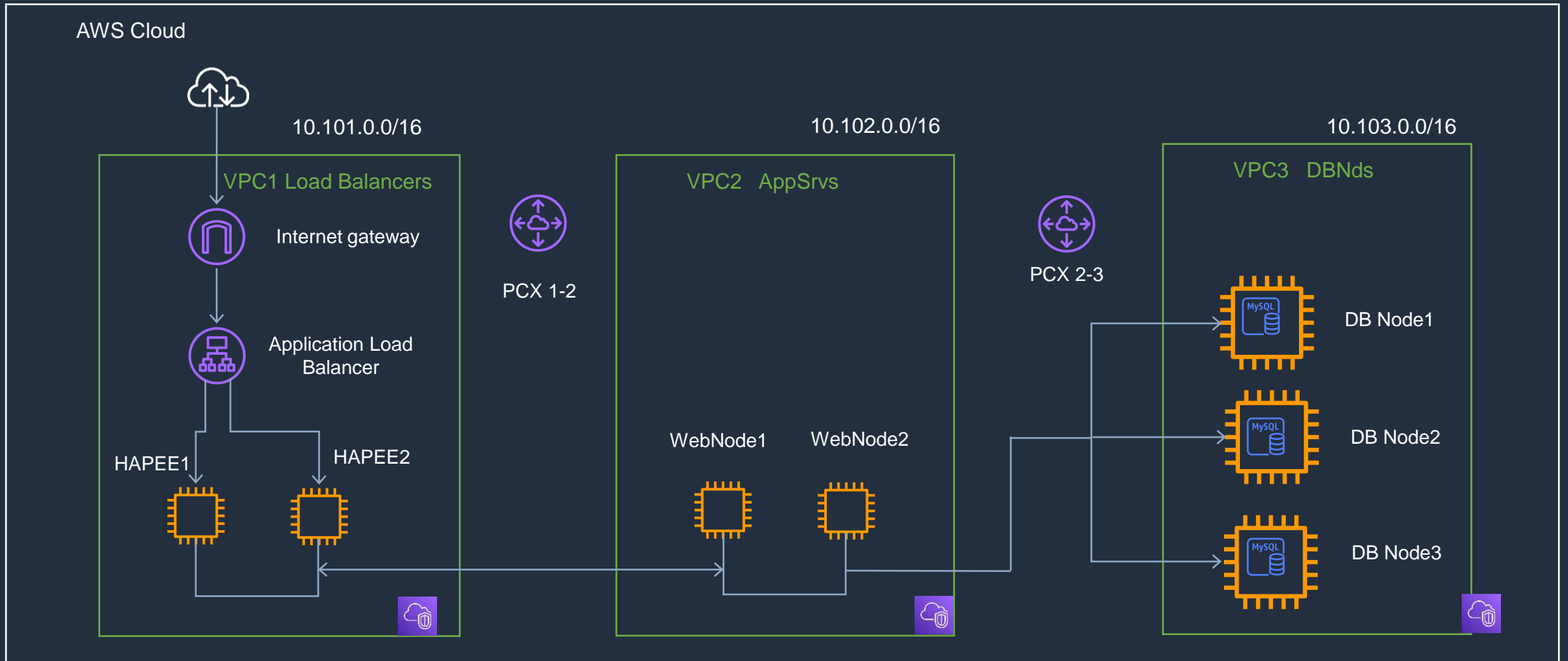
3 Different VPCs/Networks & Sec-Groups to Isolate Application from DB from Public-Access to LB

- How can this be put into a GitHub Action Pipeline to get applied when merged?
- Is there any other way to integrate it into our AWX(Ansible Tower) infrastructure to Reflect a IaC Approach?
- What could be the best way to distribute Traffic over 2 LBs or would you prefer Active/Passive? How can this be done without any interaction?
- Finally present the examples and you thoughts about the Challenges and mind-gaps in this Request?

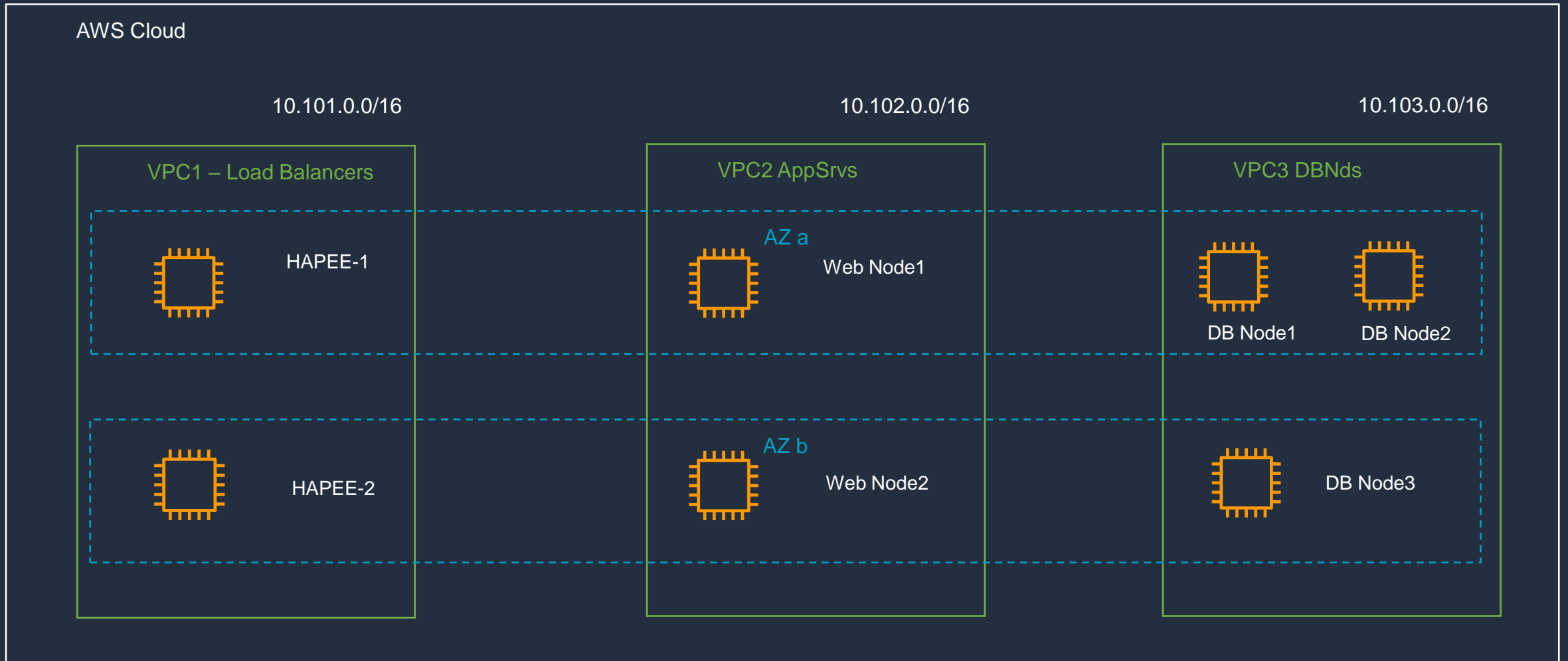
Architecture

→ Architecture that was applied

AWS Small Environment Architecture



AWS High Availability Multi AZ

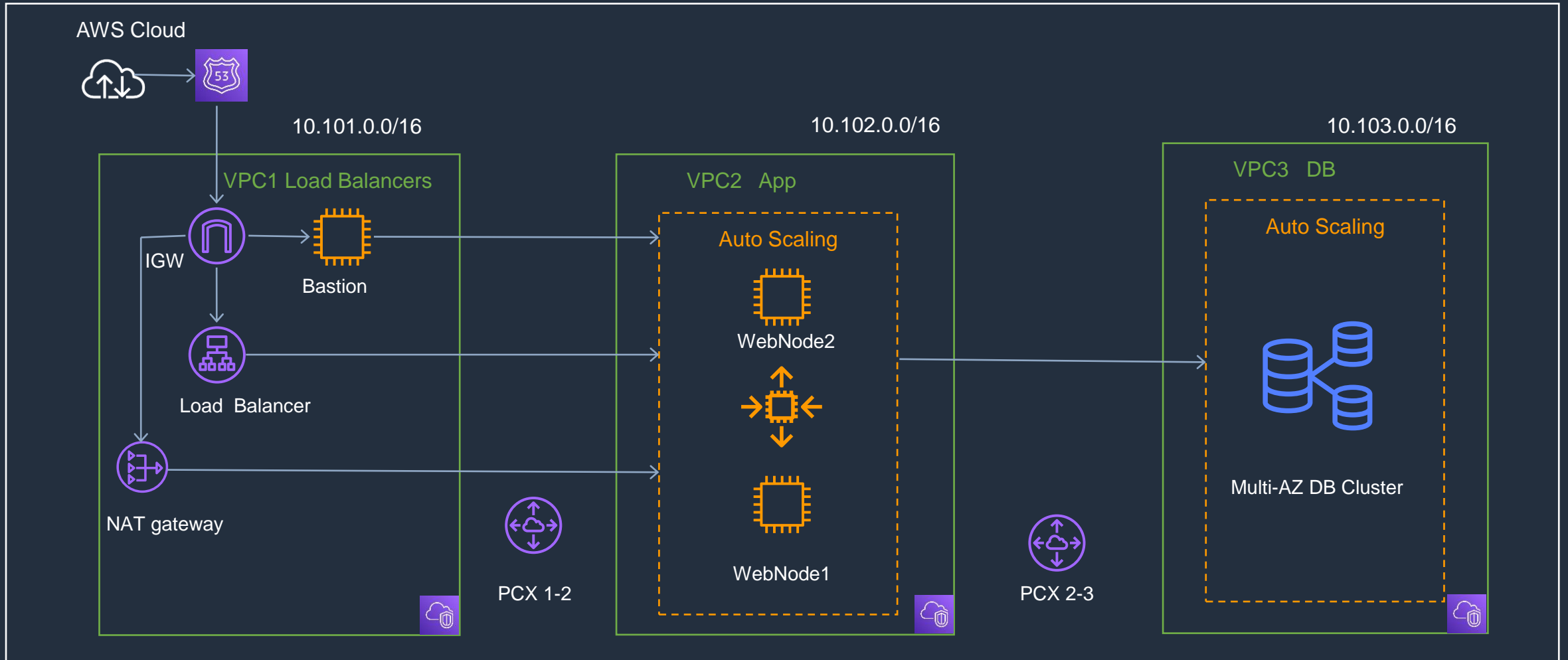


AWS High Availability Multi AZ

AWS Cloud

SmallEnvironment.tf	Output.tf	variables.tf	terraform.tfvars	Network.tf
<pre>1 #----- 2 # Example of a small environment 3 # 4 # 2 LBs (Custom Hardened Linux AMI) 5 # 2 Application-Server (default AWS AMI) 6 # 3 DB-Nodes (default AWS AMI) 7 # 3 VPCs/Networks & Sec-Groups to Isolate Application from DB from Public-Access to LB 8 # 9 # Please, custome setup here 10 # 11 # terraform.tfvars 12 #----- 13 14 #region = "us-west-2" # dev zone 15 region = "eu-central-1" # prod zone 16 instance_type = "t2.micro" 17 lb_allowed_ports = ["22", "80"] 18 app_allowed_ports = ["22", "80", "8080", "443"] 19 key_name = "MaxKeyPair" 20 app_servers_count = 2 21 hapee_lb_count = 2 22 db_nodes_count = 3 23 aws_az_count = 2 24</pre>				

AWS Small Environment Architecture



The Solution

→ Development decisions and considerations

Small environment: LBs hardening

<https://github.com/nozaq/amazon-linux-cis>

Bootstrap script for Amazon Linux to comply with [CIS Amazon Linux Benchmark](#)

Securing Amazon Linux

An objective, consensus-driven security guideline for the Amazon Linux Operating Systems.

A step-by-step checklist to secure Amazon Linux:

DOWNLOAD LATEST CIS BENCHMARK
FREE TO EVERYONE →

For Amazon Linux 2STIG (CIS Amazon Linux 2 STIG Benchmark)



LBs hardening - 149 security policies remediated



Ensure sticky bit is set on all world - writable directories

Ensure ntp is configured

Ensure time synchronization is in use

Ensure chrony is configured

Ensure X Window System is not installed

Ensure mail transfer agent is configured for local - only mode

Set Network Parameters(Host Only)

Set Network Parameters(Host and Router)

LBs hardening - 149 security policies remediated



Ensure IPv6 settings disabled

Ensure TCP Wrappers is installed

Ensure /etc/hosts.allow is configured

Ensure permissions on /etc/hosts.allow are configured

Ensure permissions on /etc/hosts.deny are configured

Uncommon Network Protocols are disabled (dccp, sctp, rds, tipc)

Firewall Configuration reviewed and updated

Reconfigured rsyslog settings

LBs hardening - 149 security policies remediated



Ensure permissions on all logfiles are configured

Ensure cron daemon is enabled

Ensure permissions on /etc/ssh/sshd_config are configured

PAM extensively reconfigured

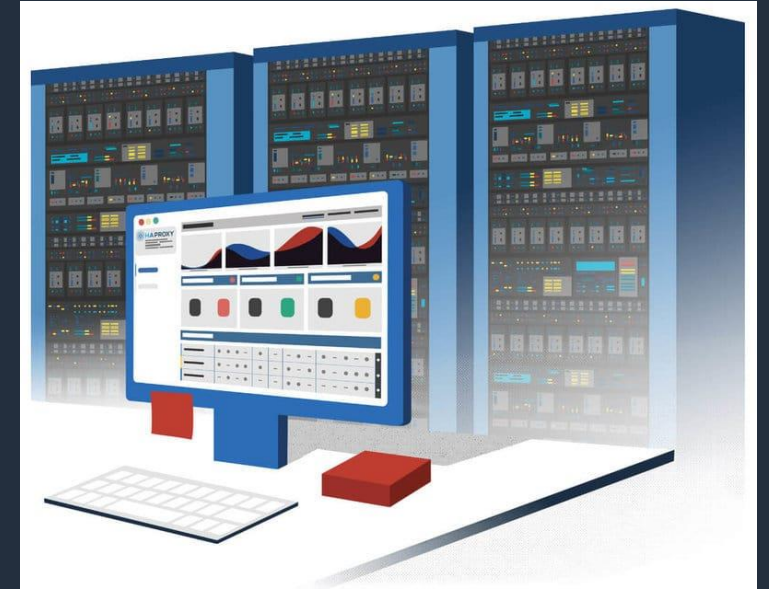
Set Shadow Password Suite Parameters

Ensure that access to the su command is restricted

Ensure that Amazon Time Sync Service is available

Small environment: HA Proxy Load Balancer

One of the popular balancers in the market, provides high availability, proxy, TCP/HTTP load-balancing. HAProxy is used by some of the reputed brands in the world, like below.



Small environment: HA Proxy Load Balancer. haproxy.cfg

Global

```
log      127.0.0.1 local2

chroot   /var/lib/haproxy

pidfile  /var/run/haproxy.pid

maxconn  4000

user     haproxy

group    haproxy

daemon
```

```
defaults

mode     http

log       global

option    httplog

option    dontlognull

option    http-server-close

option    forwardfor except 127.0.0.0/8

option    redispatch

retries   3

timeout  http-request 10s

timeout  queue       1m

timeout  connect      10s

timeout  client       1m

timeout  server       1m

timeout  http-keep-alive 10s

timeout  check        10s

maxconn  3000
```

```
frontend http_front

    bind *:80

    stats uri /stats

    default_backend http_back

#-----

# round robin balancing
# between the various backends

#-----

backend http_back

    balance roundrobin

    ${serverlist}
```

Small environment: HA Proxy. haproxy.cfg.rendered

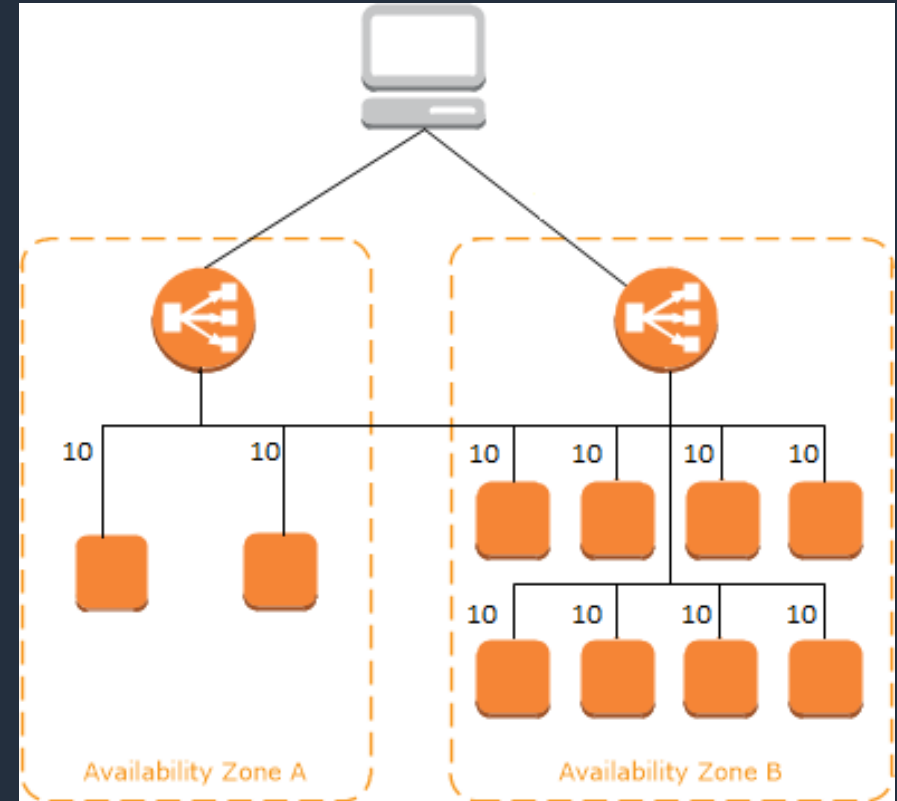
```
data "template_file" "hapee-userdata" {  
  
  template = file("hapee-userdata.sh.tpl")  
  
  vars = {  
  
    serverlist = join("\n", formatlist("  server app-%v %v:80 cookie app-%v check", aws_instance.web_node.*.id,  
aws_instance.web_node.*.private_ip, aws_instance.web_node.*.id))  
  
  }  
  
}
```

Amazon Application Load Balancer

An Application Load Balancer functions at the application layer, the seventh layer of the OSI model. After the load balancer receives a request, it evaluates the rules and then selects a target from the target group.

The default routing algorithm is round robin.

Health checks are used to monitor the health of the registered targets so that the load balancer can send requests only to the healthy targets.



Code

→ Playbook structure and code review

<https://github.com/mcrucovschii/SmallEnvironment>

- Main.tf – main Terraform file, no provisioning there
- SmallEnvironment.tf - instances declaration
- Network.tf – VPCs, subnets, security groups, gateway, route tables, ALB, DNS
- terraform.tfvars – settings and adjustments
- variables.tf – declaration of variables
- Output.tf – CLI output
- Crucovschii-AWS-SmallEnvironment.pptx (.pdf) – this presentation

Output.tf

output "data_aws_caller_identity"

output "data_aws_regions_name"

output "aws_availability_zones"

output "Web_node_public_IPs"

output "HAPEE_nodes_public_IPs"

output "LB_HAPEE_DNS_address"

terraform.tfvars

region = region where infrastructure will be deployed

instance_type = instance_type that will be used for hapee, web nodes and db nodes"

key_name = a key pair in the region where you are going to deploy

app_servers_count = a number of app servers (web nodes) to be launched

hapee_lb_count = a number of hapee balancers to be launched

db_nodes_count = a number of db nodes to be launched

aws_az_count = a number of availability zones where infrastructure will be deployed

SmallEnvironment.tf

```
resource "aws_instance" "db_node"
```

```
resource "aws_instance" "web_node"
```

```
resource "aws_instance" "hapee_node"
```


Network.tf

```
resource "aws_instance" "db_node"
```

```
resource "aws_instance" "web_node"
```

```
resource "aws_instance" "hapee_node"
```

Questions

→ Additional questions

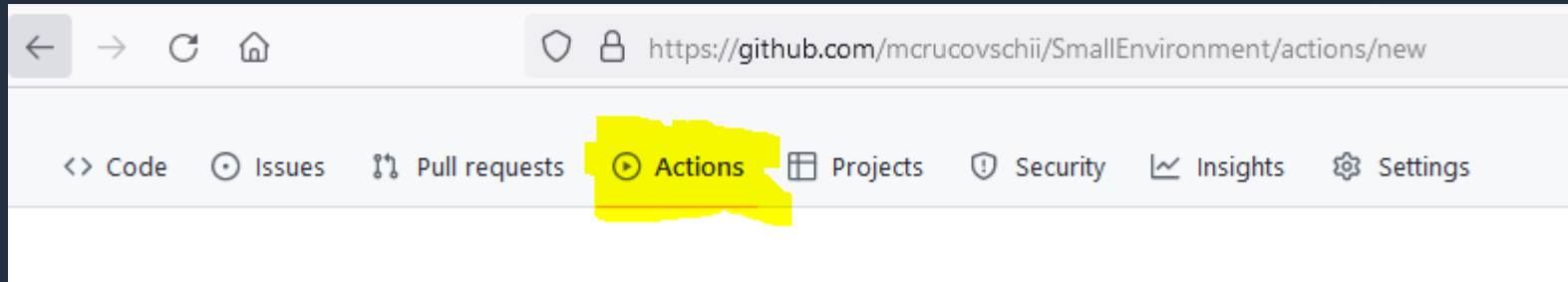
GitHub Action Pipeline

How can this be put into a GitHub Action Pipeline to get applied when merged?

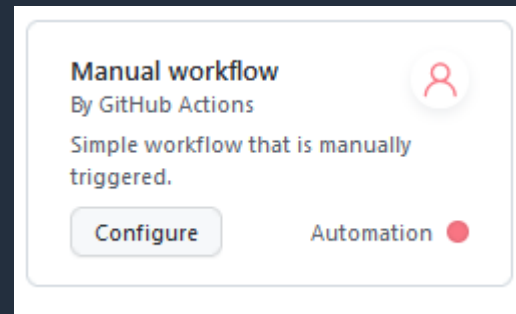
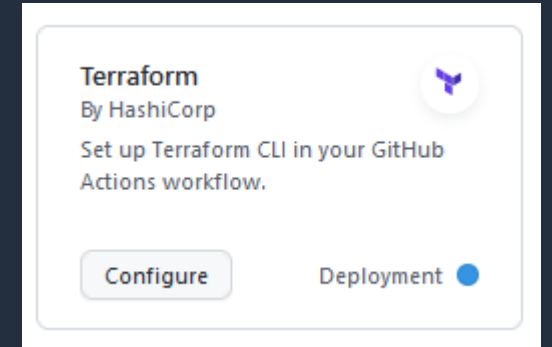
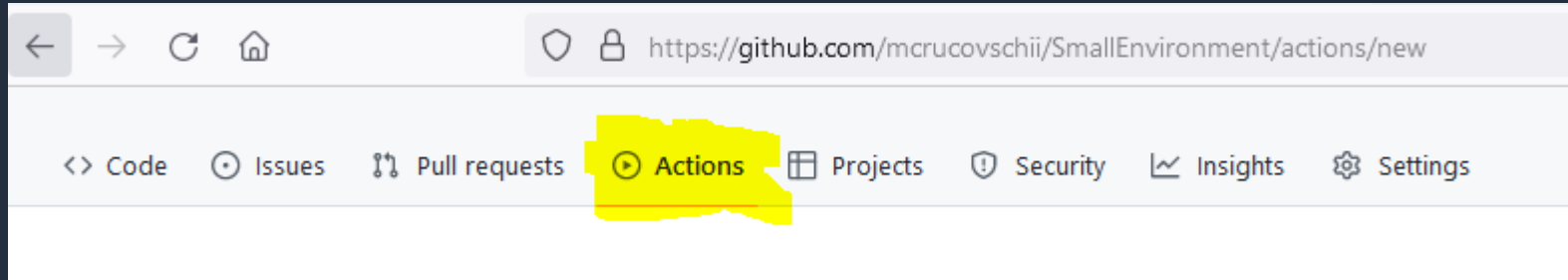


GitHub Action Pipeline

Webhooks allow external services to be notified when certain events happen. When the specified events happen, we'll send a POST request to each of the URLs you provide. Learn more in our



GitHub Action Pipeline



GitHub Action Pipeline. terraform.yml

```
on: push:  branches:  - "Master"
```

```
jobs: terraform:  name: 'Terraform'  runs-on: ubuntu-latest  environment: production
```

```
uses: hashicorp/setup-terraform@v1
```

```
    with:      cli_config_credentials_token: ${{ secrets.TF_API_TOKEN }}
```

```
    run: terraform init
```

```
    run: terraform fmt -check
```

```
    run: terraform plan -input=false
```

```
    if: github.ref == 'refs/heads/"Master"' && github.event_name == 'push'
```

```
        run: terraform apply -auto-approve -input=false
```

AWX / Ansible Tower integration

Is there way to integrate it into our AWX(Ansible Tower) infrastructure to Reflect a IaasC Approach?

?

integration into our AWX(Ansible Tower) infrastructure

AWX inventory can automatically receive the stack:

- Configure terraform output
- Ansible Tower provides webhook integration with GitHub
(<https://docs.ansible.com/ansible-tower/latest/html/userguide/webhooks.html>)
- <https://github.com/adammck/terraform-inventory>
- <https://github.com/mcrucovschii/terraform-null-ansible>
- <https://github.com/opencredo/k8s-terraform-ansible-sample/tree/master/terraform>

LB Traffic distribution

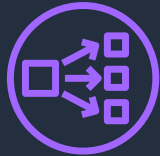
What could be the best way to distribute Traffic over 2 LBs or would you prefer Active/Passive? How can this be done without any interaction?



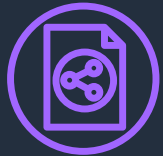
The best way to distribute Traffic over 2 LBs or would you prefer Active/Passive? How can this be done without any interaction?



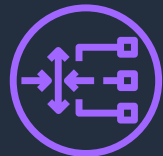
Application Load Balancer



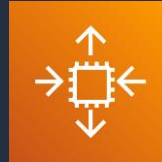
Network Load Balancer



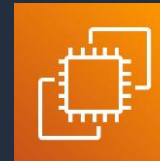
Classic Load Balancer



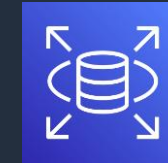
Gateway Load Balancer



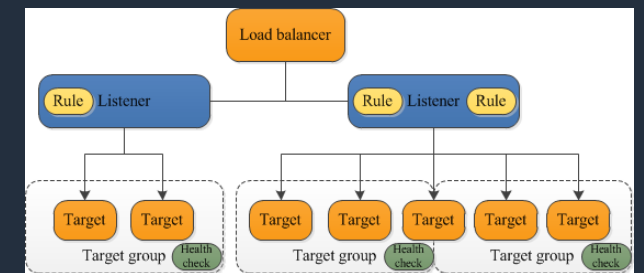
Auto Scaling



Amazon EC2



Amazon Relational Database Service (Amazon RDS)



The best way to distribute Traffic over 2 LBs or would you prefer Active/Passive? How can this be done without any interaction?

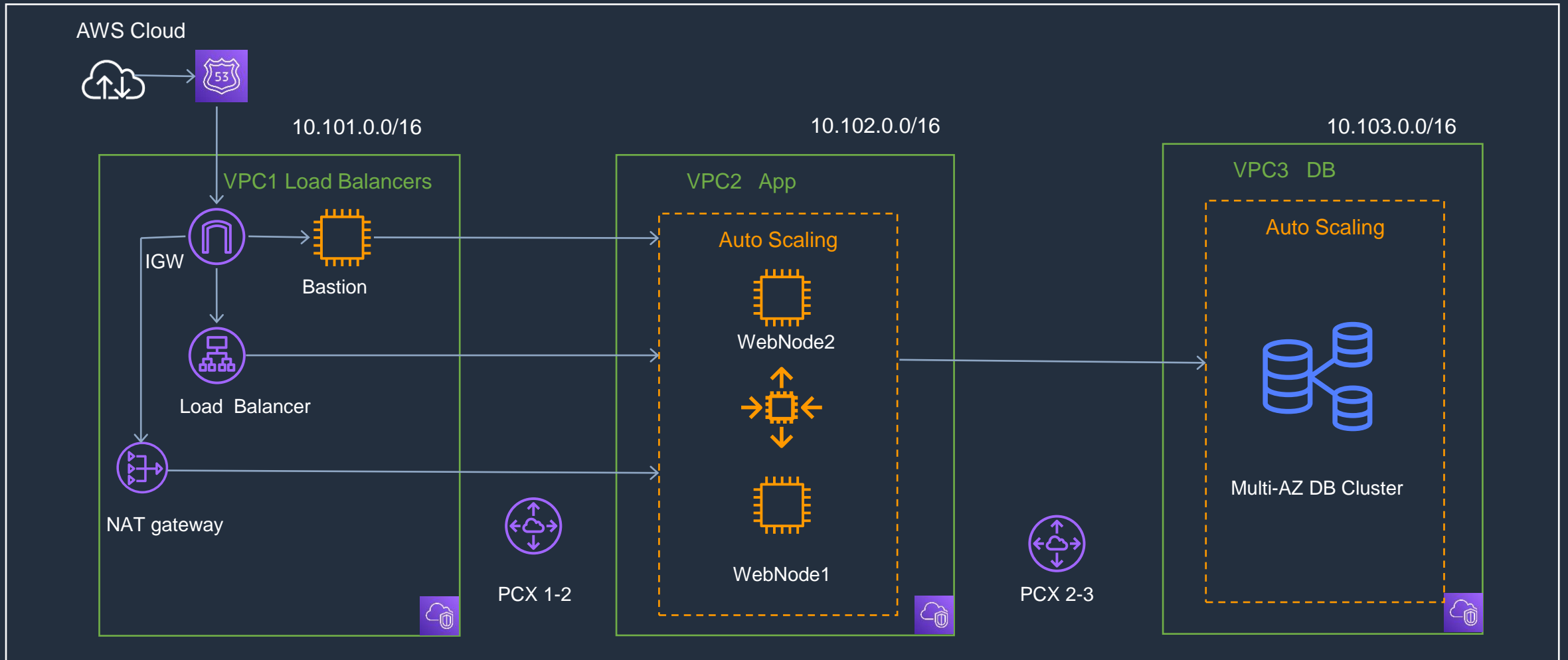
- In an **active-passive** configuration, the load balancer recognizes a failed node and redirects traffic to the next available node.
- In an **active-active** configuration, the load balancer spreads out the workload's traffic among multiple nodes.

My thoughts / My vision

**Present the examples and you thoughts about the
Challenges and mind-gaps in this Request?**



My thoughts / My vision



Questions ?

