The 1 Infrastructure Upgrades

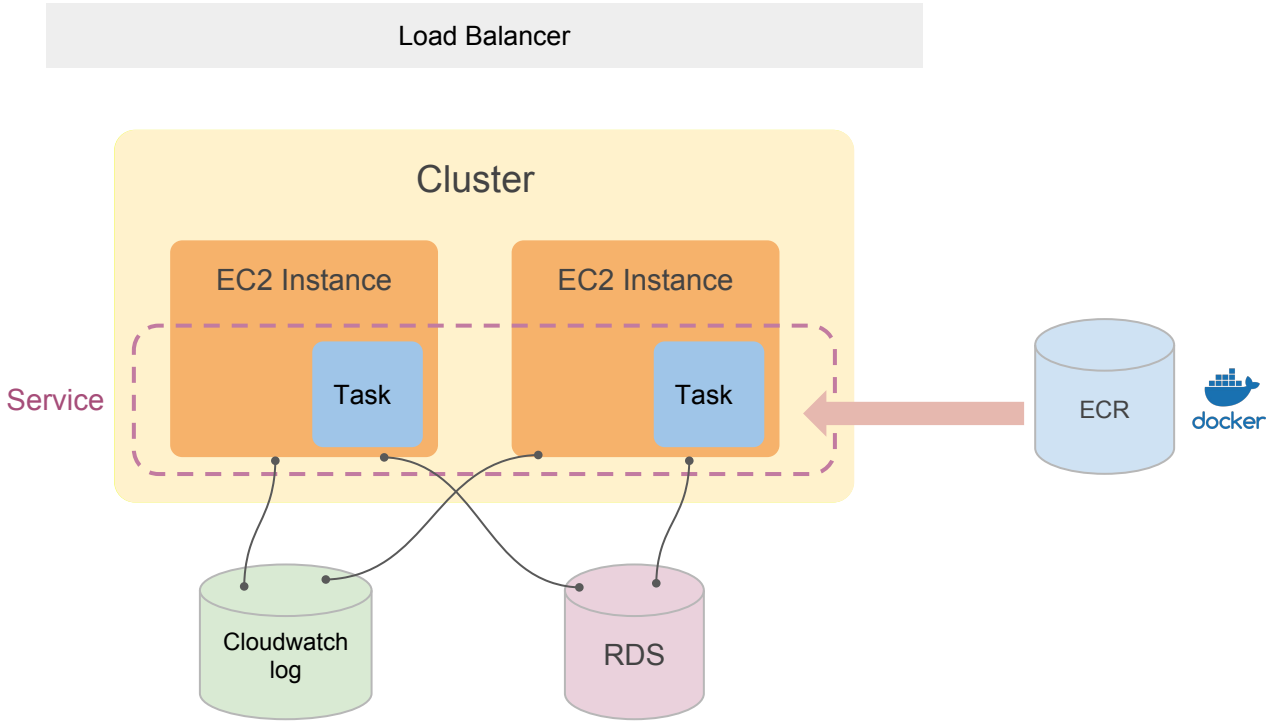
Ankit, Ros

Growth Session #21 - January 24-25 2019

Goals

- Setup the Terraform for The 1
- Implement caching using Varnish to improve the performance

The1 Infrastructure



Current Setup

1. Create cluster and EC2 instances with ECS CLI


Manually create the load balancer


Manually create a cloudwatch log group


Manually create and upload the image to the registry

5. Create and start the cluster service with ECS CLI

Applying terraform

```
resource "aws_ecr_repository" "repository" {
  name = "the1-web"

  provisioner "local-exec" {
    command = "echo ${aws_ecr_repository.repository.repository_url} >> .repository"
  }
}
```

```
resource "aws_cloudwatch_log_group" "log" {
  name = "${local.name}"
  retention_in_days = 7

  tags = {
    Name = "${local.name}"
  }
}
```

```
resource "aws_elb" "load_balancer" {
  name                = "${local.name}"
  internal            = false
  security_groups     = ["${var.security_group_id}"]
  subnets             = ["${var.subnet_id_1}", "${var.subnet_id_2}"]
  connection_draining = false

  listener {
    instance_port     = 80
    instance_protocol = "http"
    lb_port           = 80
    lb_protocol       = "http"
  }

  listener {
    instance_port      = 8080
    instance_protocol  = "http"
    lb_port            = 8080
    lb_protocol        = "http"
  }

  tags = {
    Name = "${local.name}"
  }
}
```

Improvement

1. Create cluster and EC2 instances with ECS CLI

Terraform apply

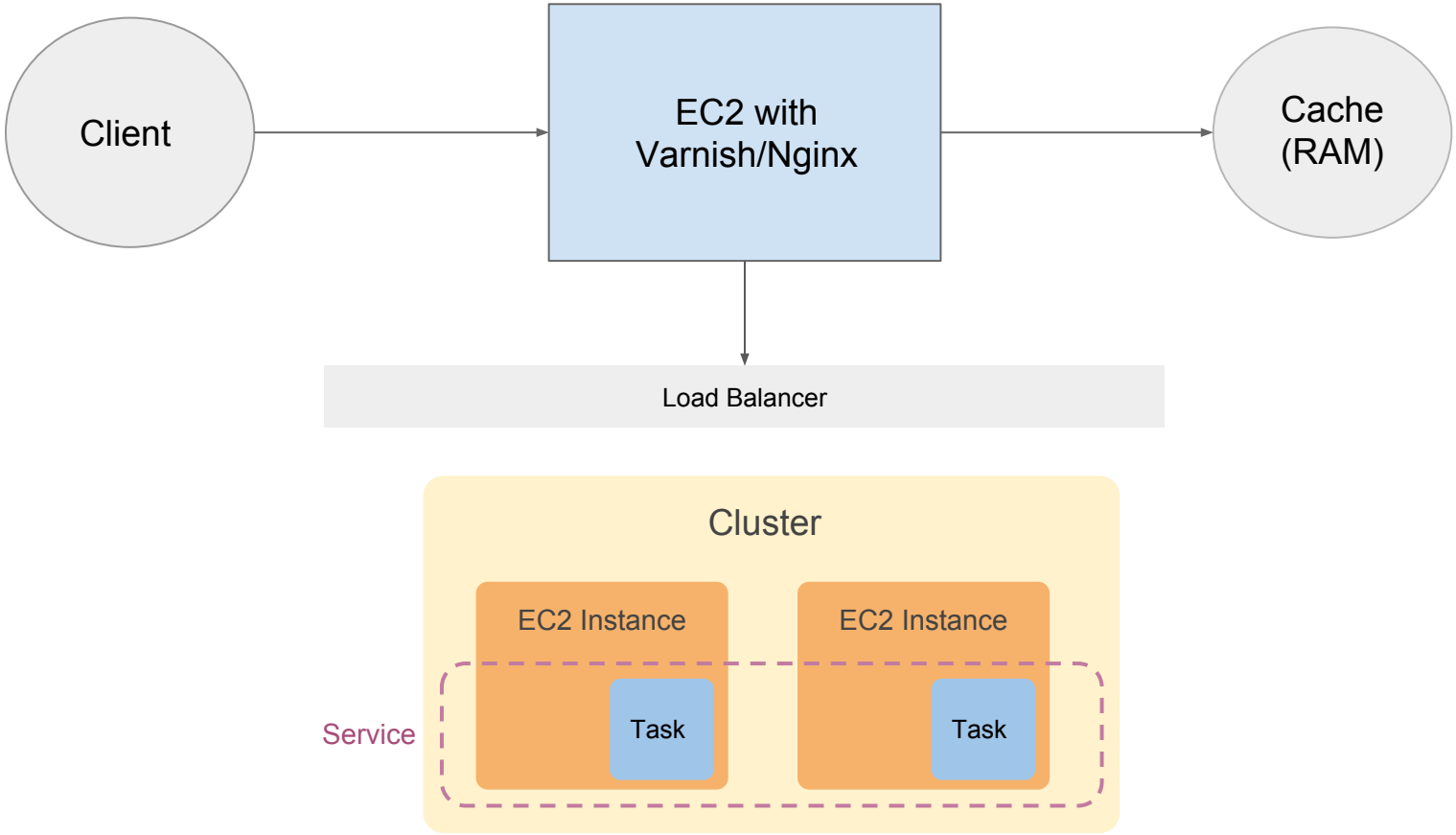
5. Create and start the cluster service with ECS CLI

Next Stepts

- Move the database and keypair to terraform
- Remove the ecs cli steps

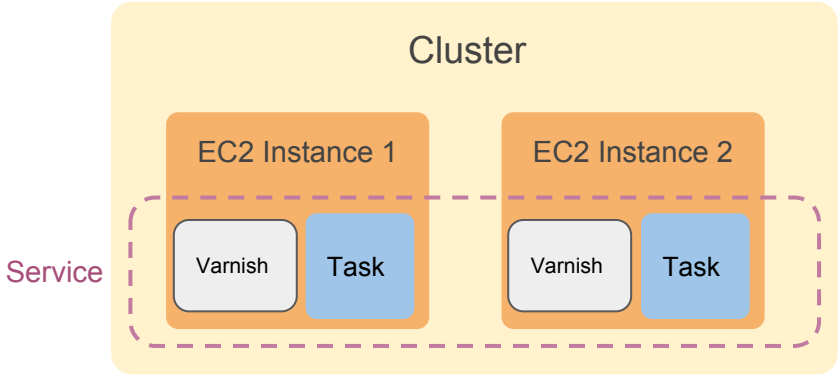
```
→  terraform git:(chore/terraform) ✗ ecs-cli up --keypair the1-web-tf --capability-iam --size 1 --instance-t
ype t2.small --force
INFO[0000] Using recommended Amazon Linux 2 AMI with ECS Agent 1.25.0 and Docker version 18.06.1-ce
INFO[0000] Created cluster                               cluster=the1-web-tf region=ap-southeast-1
INFO[0001] Waiting for your CloudFormation stack resources to be deleted...
INFO[0001] Cloudformation stack status                   stackStatus=DELETE_IN_PROGRESS
INFO[0062] Waiting for your cluster resources to be created...
INFO[0062] Cloudformation stack status                   stackStatus=CREATE_IN_PROGRESS
INFO[0123] Cloudformation stack status                   stackStatus=CREATE_IN_PROGRESS
INFO[0183] Cloudformation stack status                   stackStatus=CREATE_IN_PROGRESS
VPC created: vpc-0d52cffcec655a897
Security Group created: sg-09f34fb61fbdfd9f3
Subnet created: subnet-06afb387bd53274ec
Subnet created: subnet-0248ddfbd43894e92
Cluster creation succeeded.
→  terraform git:(chore/terraform) ✗ terraform apply
var.environment
  Enter a value: tf

var.security_group_id
  Enter a value: sg-09f34fb61fbdfd9f3

var.subnet_id_1
  Enter a value: subnet-06afb387bd53274ec

var.subnet_id_2
  Enter a value: subnet-0248ddfbd43894e92

var.vpc_id
  Enter a value: vpc-0d52cffcec655a897
```

Infrastructure with Varnish

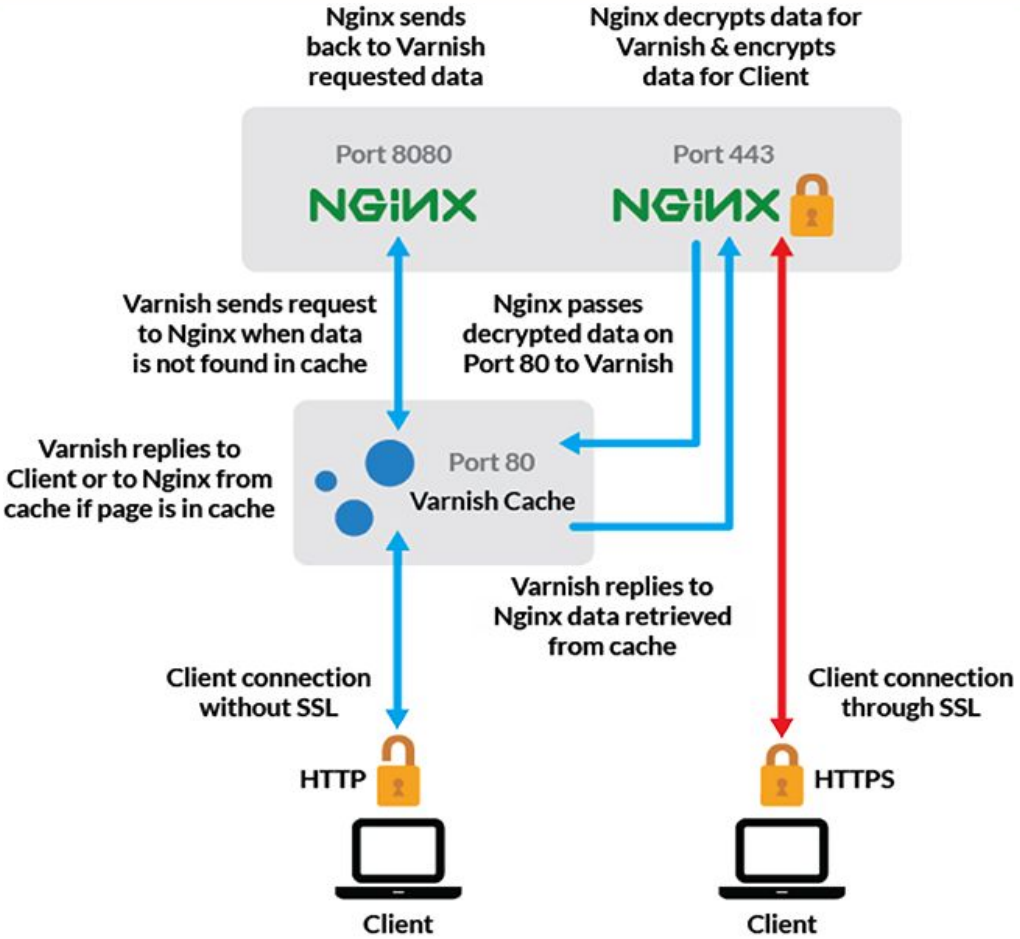


Why using separate instance?

- Cached requests do not hit the main server at all, reducing its load significantly.
- Seperate storage for cache.
- All cache gets stored at one place, this makes it easy to clear the cache.
- As the cache gets stored at one place, cache hits would be higher.



Inner-working of a Caching Instance



Setup and Config

1. Create EC2 instance, install nginx and varnish.

2. Make following changes in /etc/nginx/sites-enabled/default

```
listen 8080 default_server;
listen [::]:8080 default_server;
```

```
location / {
  proxy_pass      https://www.the1.co.th;
  proxy_redirect  https://www.the1.co.th/ /;
  proxy_read_timeout 60s;
  add_header Vary Authorization;
  # Now use this solution:
  proxy_ignore_headers Cache-Control;
  proxy_ignore_headers    Expires;

  # Or if above didn't work maybe this:
  proxy_hide_header Cache-Control;
  proxy_hide_header pragma;
  proxy_hide_header Expires;
}
```

Varnish Code and Demo

```
backend default {
    .host = "127.0.0.1";
    .port = "8080";
}

# Respond to incoming requests.
sub vcl_recv {

  if (req.method == "POST") {
    return (pass);
  }

  if (req.http.Accept-Encoding) {
    if (req.http.Accept-Encoding ~ "gzip") {
      set req.http.Accept-Encoding = "gzip";
    } else if (req.http.Accept-Encoding ~ "deflate") {
      set req.http.Accept-Encoding = "deflate";
    } else {
      unset req.http.Accept-Encoding;
    }
  }
  #Varnish <= 3.x calls this "return (lookup);"
  return (hash);


}
```

Performance Comparison on Jmeter(Uncached vs cached)

elapsed	label
967	Login - email
1046	Login - mobileNo
1278	Me
207	My Rewards
1423	Recommended Rew
8390	Redeemable Reward
10032	[Screen - Home]
3862	Discover Banner
4300	Redemption Banner
4095	Privilege Banner
3347	Exchange Banner
3001	Privilege Content
2954	Exchange Content
318	Brand Store Content
1545	Article Content
286	All Categories
3439	Discovery Category C
3375	Discovery Category C
3481	Discovery Category C
5426	Discovery Category C
3482	Discovery Category C
4605	Discovery Category C
4404	Discovery Category C
51995	[Screen - Discover]
6739	Reward Catalog
3679	Ecash Content

elapsed	label
938	Login - email
715	Login - mobileNo
32	Me
31	My Rewards
64	Recommended Rew
116	Redeemable Reward
216	[Screen - Home]
31	Discover Banner
36	Redemption Banner
38	Privilege Banner
67	Exchange Banner
42	Privilege Content
39	Exchange Content
34	Brand Store Content
546	Article Content
58	All Categories
46	Discovery Category C
34	Discovery Category C
44	Discovery Category C
44	Discovery Category C
44	Discovery Category C
43	Discovery Category C
39	Discovery Category C
1217	[Screen - Discover]
82	Reward Catalog
33	Ecash Content

- Not much difference for login requests as they cannot be cached.
- For now, we enabled caching for all endpoints irrespective of the content.

Next Steps

- Improve upon caching to make it production ready
  - Identify endpoints whose response is independent of user.
  - Create endpoint to clear cache.
  - Maybe create a service which refreshes cache of such endpoints.
- Zero downtime deployment
  - Use application load balancer with dynamic port binding
  - Blue/Green Deployment with AWS Codedeploy

Thanks!

Contact Nimble

nimblehq.co

hello@nimblehq.co

Bangkok

399 Interchange 21 Sukhumvit Road, Unit #2402-03, Klong Toei, Wattana, Bangkok 10110, Thailand

Singapore

28C Stanley St, Singapore 068737

Hong Kong

20th Floor, Central Tower

28 Queen's Road, Central, Hong Kong

nimble