

[Sign up](#)[Sign In](#)

Write



Published in Geek Culture



Jazz Tong

[Follow](#)May 14, 2021 · 8 min read · [Listen](#)

MongoDB Replica Set in AWS ECS with Terraform

Today we shall go through the solution of using ECS to set up MongoDB replica Set. Although it is possible to set up MongoDB in ECS using the Fargate launch type, we will follow the best practices in ECS for persistent workload using the EC2 launch type. To allow reproducible artifacts for this solution, we will use Terraform for all the configurations. You are welcomed to reuse this template and modify it accordingly.

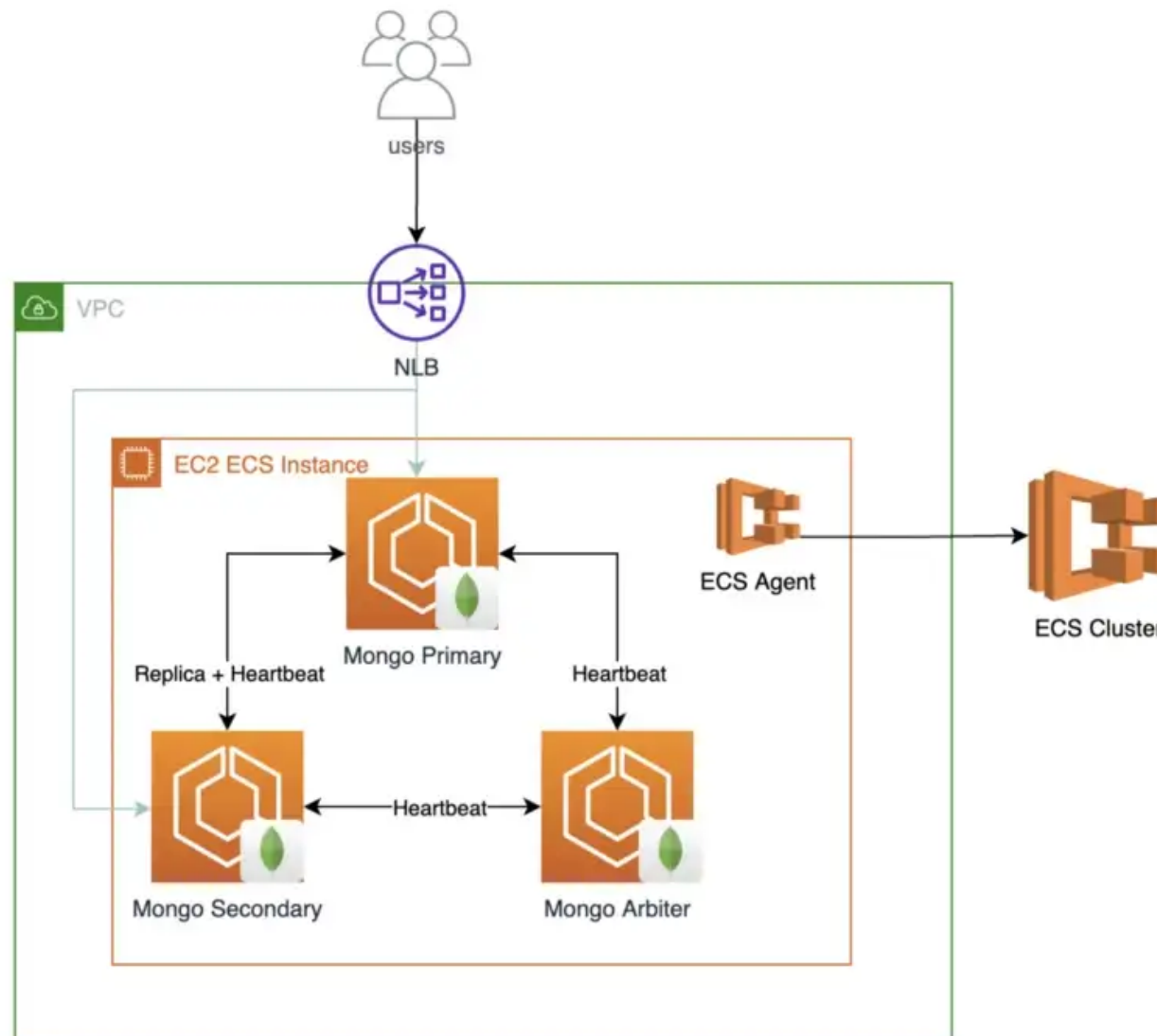
High-level diagram



19



1



As illustrated in the diagram above, we will use EC2 as compute resource for our ECS Cluster. For simplicity's sake, only use 1 EC2 to run 3 MongoDB nodes, but you can add multiple EC2s to maximize the redundancy. Here is what we will build:-

1. 1 EC2 using ECS optimized image for ECS Instance
2. 3 MongoDB Services using Bitnami Image
3. 1 ECS Cluster
4. 1 NLB to route traffic to Primary and Secondary MongoDB service

Why Bitnami MongoDB Image

To set up a replica set using docker, you are required to configure the replica set configuration and add nodes accordingly. MongoDB official image does not provide an out-of-the-box auto replica configuration, hence we will use the Bitnami MongoDB image, which allows users to configure the replica set using an environment variable.

Prerequisite knowledge

The solution shared in this article requires an understanding of the following knowledge:-

1. Docker container
2. AWS IAM, EC2, Network load balancer, ECS Service and Task definition
3. Terraform scripting
4. MongoDB configuration

Prerequisite Setup

You need the following tools to follow this solution:-

1. [Visual Studio Code](#)
2. [AWS CLI Install](#)
3. [AWS Session Manager Plugin](#)
4. [MongoDB Compass](#)
5. [Terraform v0.15+](#)

VPC Trucking

In this solution, we will use `awsvpc` network mode for all the MongoDB Service. Refer [here](#) to the detailed `awsvpc` network mode. `awsvpc` consumes network interface or Elastic Network Interface (ENI) of each EC2, and there is a limited ENI for each EC2 type. [VPC trunking](#) is a new feature that allows us to launch twice as much task using `awsvpc` network mode with EC2 launch type. But this is the opt-in option, you need to enable this by following this [configuration](#), or enable in account level setting in ECS console as below:-

AWSVPC Trunking

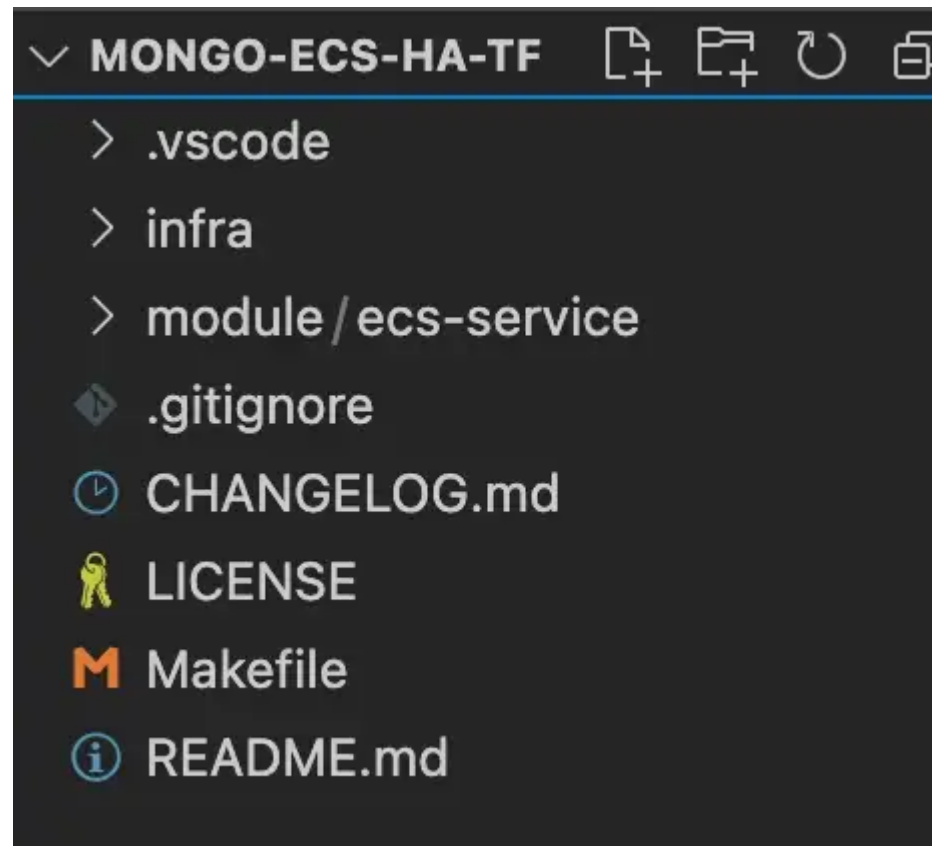
Amazon ECS ENI Trunking allows you to increase the number of elastic network interfaces per EC2 instance. For more information about the specific ENI density allowed for each instance type, see [Amazon ECS ENI Trunking](#).

Select or deselect the check box below to enable or disable trunking. A change in this setting requires a restart of the ECS service.

Resource	My IAM user or role account settings
AWSVPC Trunking	<input checked="" type="checkbox"/>

Initialize project

As the project is comprehensive to go through step by step, we will use the finished project for the explanation. Check out the [source](#) here and configure AWS credentials. The project structure looks like this:-



ECS Service module

Inside the project, there is one module that allows users to provision ECS service with a predefined setting. We will go through the important resource for the module.

aws_ecs_service resource

This is the **ECS** service resource that provision ECS service. We configure it to use **EC2** launch type by default and allows control of **load_balancer** settings. As MongoDB node cannot run more than one task at a time, we configure `deployment_maximum_percent` it to 100. As a result, any new deployment will immediately stop the existing task.

```
1 resource "aws_ecs_service" "this" {
2     name          = var.name
3     cluster       = var.cluster
4     launch_type   = "EC2"
5
6     task_definition = aws_ecs_task_definition.this.arn
7     force_new_deployment = false
8     desired_count    = var.desired_count
9
10    dynamic "load_balancer" {
11        for_each = var.create_lb ? aws_lb_target_group.this : []
12        content {
13            target_group_arn = load_balancer.value.arn
14            container_name   = var.name
15            container_port   = var.containerPort
16        }
17    }
18
19    network_configuration {
20        subnets          = var.subnets
21        security_groups   = var.security_groups
22        assign_public_ip = false
23    }
24
25    service_registries {
26        registry_arn = aws_service_discovery_service.this.arn
27    }
28
29    # Service only run one instance at a time
30    deployment_maximum_percent    = 100
31    deployment_minimum_healthy_percent = 0
32 }
```

module_ecs-service_service.tf hosted with ❤ by GitHub

[view raw](#)

aws_ecs_task_definition resource

We need task definition to control how our container run in ECS. The following resource allows dynamic volume configuration, where it is an important part to persist MongoDB data. To manage the volume using [docker volume](#), we add `docker_volume_configuration` section to allow us to configure local or other volume drivers. The `placementConstraints` configuration also provides to allow us to control where our container/task should provision in which EC2 instance. You can create 3 different EC2 with different labels and ensure they launch in respective AZ and EC2.

```
1  resource "aws_ecs_task_definition" "this" {
2      family                = var.name
3      memory                = var.memory
4      requires_compatibilities = ["EC2"]
5      task_role_arn          = var.task_role_arn
6      execution_role_arn      = var.execution_role_arn
7      network_mode            = "awsvpc"
8
9
10     dynamic "volume" {
11         for_each = var.volumes == null ? [] : var.volumes
12         content {
13             name          = volume.value.name
14             host_path      = try(volume.value.host_path, null)
15             dynamic "docker_volume_configuration" {
16                 for_each = can(volume.value.docker_volume_configuration) ? [try(volume.value.docker_volu
17                 content {
18                     scope          = try(docker_volume_configuration.value.scope, null)
19                     autoprovision = try(docker_volume_configuration.value.autoprovision, null)
20                     driver          = try(docker_volume_configuration.value.driver, null)
21                 }
22             }
23         }
24     }
25
26     container_definitions = jsonencode(
27         [{
28             "name" : var.name
29             "image" : var.image,
30             "portMappings" : [
31                 { containerPort = var.containerPort }
32             ],
33             "environment" : var.environment,
34             "mountPoints" : var.mountPoints
```

```
34     "placementConstraints" : var.placementConstraints,
35     "placementConstraints" : var.placementConstraints,
36     "volumes" : var.volumes,
37     "logConfiguration" : {
38       "logDriver" : "awslogs",
39       "options" : {
40         "awslogs-create-group" : "true",
41         "awslogs-group" : "/ecs/${var.name}",
42         "awslogs-region" : data.aws_region.this.name,
43         "awslogs-stream-prefix" : "ecs"
44       }
45     },
46   }]
47 )
48 }
```

ecs_module_task_def.tf hosted with ❤ by GitHub

[view raw](#)

```
1  #! /bin/bash
2  set -e
3  # Ouput all log
4  exec > >(tee /var/log/user-data.log|logger -t user-data -s 2>/dev/console) 2>&1
5
6  # Set ECS agent setting
7  cat <<'EOF' >> /etc/ecs/ecs.config
8  ECS_CLUSTER=${ECS_CLUSTER}
9  ECS_INSTANCE_ATTRIBUTES={"mongo": "primary"}
10 ECS_CONTAINER_STOP_TIMEOUT=2s
11 ECS_IMAGE_PULL_BEHAVIOR=prefer-cached
12 EOF
13
14 # Enable password authentication
15 sed 's/PasswordAuthentication no/PasswordAuthentication yes/' -i /etc/ssh/sshd_config
16 systemctl restart sshd
17 service sshd restart
18
19 # Add ecs-user
20 useradd ecs-user
21 usermod -aG wheel ecs-user
22 echo "${ECS_USER_PASSWORD}" | passwd --stdin ecs-user
23
24 # Report end
25 echo 'Done Initialization'
```

ecs_ha_tf_user_data.tmpl.sh hosted with ❤ by GitHub

[view raw](#)

mongo_primary resource

We will create a MongoDB node using `ecs-service` module. In this Mongo Primary node, we configure Network Load Balancer , Bitnami Mongo replica setting using an environment variable, volume and mount point to persist the data. Refer here for [Bitnami Mongo Image configuration](#).

desired_count must set to 1 for successful provision, placementConstraints must also be configured to match EC2 label

```
1  module "mongo_primary" {
2      source = "../module/ecs-service"
3
4      cluster      = aws_ecs_cluster.this.id
5      name         = "${var.app_id}-primary"
6      image        = var.image
7      containerPort = 27017
8
9      create_lb     = var.nlb_enabled
10     lb_arn        = var.nlb_enabled ? aws_lb.this[0].arn : ""
11     listener_port = 27017
12
13     desired_count = var.primary_enabled ? 1 : 0
14     memory        = var.memory
15     environment = [
16         { "name" : "MONGODB_ROOT_PASSWORD", "value" : var.mongo_password },
17         { "name" : "MONGODB_ADVERTISED_HOSTNAME", "value" : "mongo-ecs-primary.ecs.demo" },
18         { "name" : "MONGODB_REPLICA_SET_MODE", "value" : "primary" },
19         { "name" : "MONGODB_REPLICA_SET_KEY", "value" : "replicasetkey123" }
20     ]
21     volumes = [
22         {
23             "name" : "primary-data",
24             "docker_volume_configuration" : {
25                 "scope" : "shared",
26                 "autoprovision" : true,
27                 "driver" : "local"
28             }
29         }
30     ]
31     mountPoints = [
32         {
33             "containerPath" : "/bitnami/mongodb",
34             "sourceVolume" : "primary-data"
```

```
34     sourceVolume = primary_data
35   }
36 ]
37 placementConstraints = [
38   {
39     "expression" : "attribute:mongo == primary",
40     "type" : "memberOf"
41   }
42 ]
43 discovery_namespace_id = aws_service_discovery_private_dns_namespace.this.id
44 security_groups         = [aws_security_group.this.id]
45 subnets                = data.aws_subnet_ids.this.ids
46 task_role_arn           = aws_iam_role.task.arn
47 execution_role_arn      = aws_iam_role.task.arn
48
49 depends_on = [
50   aws_lb.this
51 ]
52 }
```

mongo_ecs_ha_mongo-primary.tf hosted with ❤️ by GitHub

[view raw](#)

```
1  module "mongo_secondary" {
2      source = "../module/ecs-service"
3
4      cluster      = aws_ecs_cluster.this.id
5      name         = "${var.app_id}-secondary"
6      image        = var.image
7      containerPort = 27017
8
9      create_lb    = var.nlb_enabled
10     lb_arn       = var.nlb_enabled ? aws_lb.this[0].arn : ""
11     listener_port = 27018
12
13     desired_count = var.secondary_enabled ? 1 : 0
14     memory        = var.memory
15     environment = [
16         { "name" : "MONGODB_ADVERTISED_HOSTNAME", "value" : "mongo-ecs-secondary.ecs.demo" },
17         { "name" : "MONGODB_REPLICA_SET_MODE", "value" : "secondary" },
18         { "name" : "MONGODB_INITIAL_PRIMARY_HOST", "value" : "mongo-ecs-primary.ecs.demo" },
19         { "name" : "MONGODB_INITIAL_PRIMARY_ROOT_PASSWORD", "value" : var.mongo_password },
20         { "name" : "MONGODB_REPLICA_SET_KEY", "value" : "replicasetkey123" }
21     ]
22     placementConstraints = [
23         {
24             "expression" : "attribute:mongo == primary",
25             "type" : "memberOf"
26         }
27     ]
28     discovery_namespace_id = aws_service_discovery_private_dns_namespace.this.id
29     security_groups        = [aws_security_group.this.id]
30     subnets               = data.aws_subnet_ids.this.ids
31     task_role_arn          = aws_iam_role.task.arn
32     execution_role_arn     = aws_iam_role.task.arn
33
34     depends_on = [
```



```
1  module "mongo_arbiter" {
2      source = "../module/ecs-service"
3
4      cluster      = aws_ecs_cluster.this.id
5      name         = "${var.app_id}-arbiter"
6      image        = var.image
7      containerPort = 27017
8
9      desired_count = var.arbiter_enabled ? 1 : 0
10     memory        = var.memory
11     environment = [
12         { "name" : "MONGODB_ADVERTISED_HOSTNAME", "value" : "mongo-ecs-arbiter.ecs.demo" },
13         { "name" : "MONGODB_REPLICA_SET_MODE", "value" : "arbiter" },
14         { "name" : "MONGODB_INITIAL_PRIMARY_HOST", "value" : "mongo-ecs-primary.ecs.demo" },
15         { "name" : "MONGODB_INITIAL_PRIMARY_ROOT_PASSWORD", "value" : var.mongo_password },
16         { "name" : "MONGODB_REPLICA_SET_KEY", "value" : "replicasetkey123" }
17     ]
18     placementConstraints = [
19         {
20             "expression" : "attribute:mongo == primary",
21             "type" : "memberOf"
22         }
23     ]
24     discovery_namespace_id = aws_service_discovery_private_dns_namespace.this.id
25     security_groups        = [aws_security_group.this.id]
26     subnets               = data.aws_subnet_ids.this.ids
27     task_role_arn          = aws_iam_role.task.arn
28     execution_role_arn     = aws_iam_role.task.arn
29
30     depends_on = [
31         aws_lb.this
32     ]
33 }
```

Start provision the solution

We already walk through some of the important setups in this solution, let's start provision the solution. Open the checkout source in Visual Studio Code.

We will use Makefile for all the configuration. You can use the native script in the Makefile to run if your machine does not support Makefile

Make sure you configure aws credential before start the provisioning

terraform init

Run `make init` in the project root to initialize the Terraform project.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

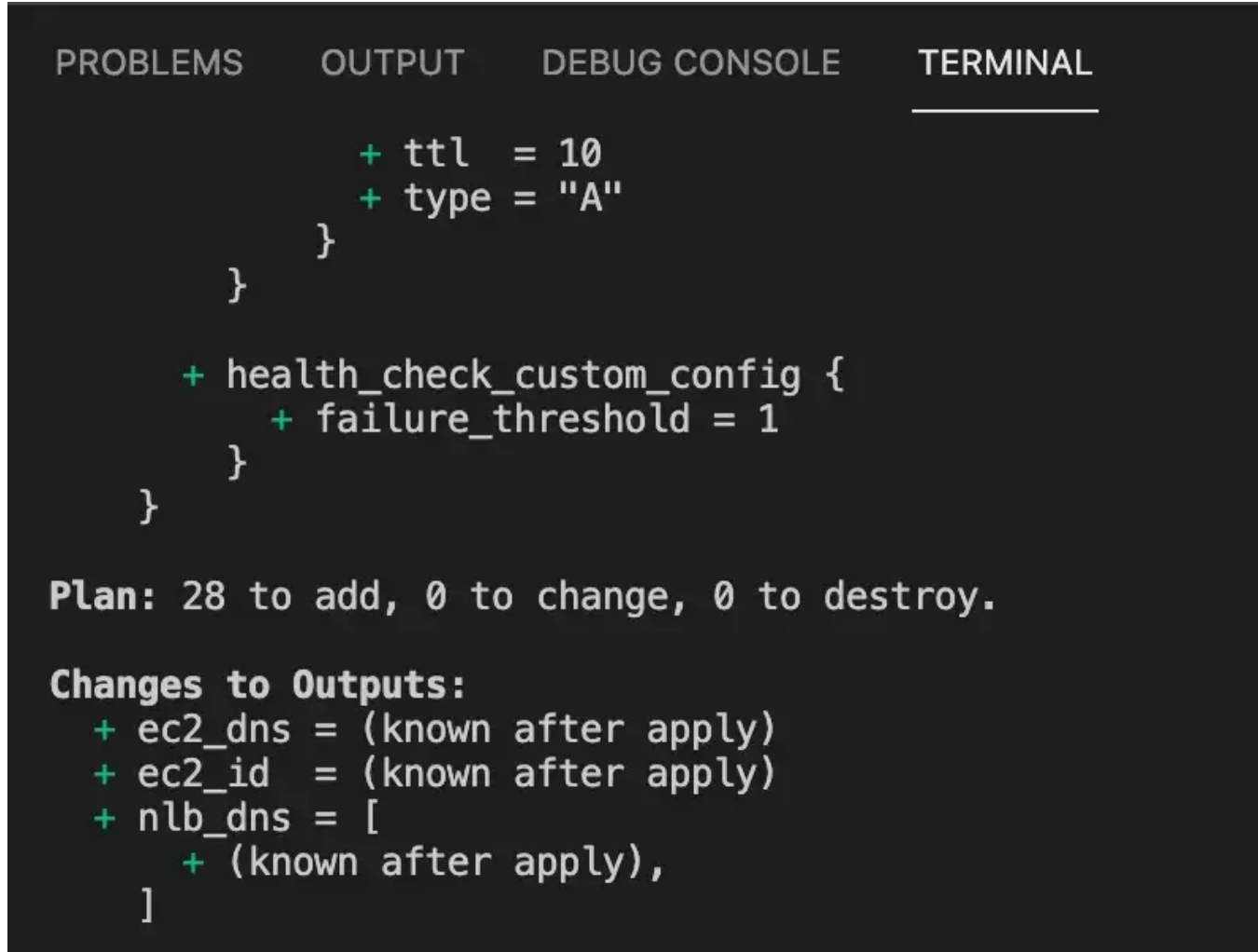
- Reusing previous version of hashicorp/template from the dependency lock file
- Using previously-installed hashicorp/aws v3.37.0
- Using previously-installed hashicorp/template v2.2.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

terraform plan

Run `make plan` in the project root to verify the Terraform plan.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

      + ttl      = 10
      + type     = "A"
    }
  }

+ health_check_custom_config {
+   failure_threshold = 1
+ }
}

Plan: 28 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ ec2_dns = (known after apply)
+ ec2_id  = (known after apply)
+ nlb_dns = [
+   (known after apply),
+ ]
```

terraform apply

After Terraform plan success, we can provision it into our environment using terraform plan. Run `make apply` to provision the Terraform script.

```
ary]
module.mongo_primary.aws_ecs_service.this: Creation complete after 1s [
Apply complete! Resources: 28 added, 0 changed, 0 destroyed.

Outputs:

ec2_dns = "ec2-13-212-159-53.ap-southeast-1.compute.amazonaws.com"
ec2_id = "i-0187dfcc15f5b8c75"
nlb_dns = [
  "mongo-ecs-NLB-fa430e1368b12b18.elb.ap-southeast-1.amazonaws.com",
]
```

Explore your result

Great, you follow me. At this point, you should successfully provision the entire replica set into your account. Let's check your AWS environment to verify the result.

Explore your Cluster

Open your ECS home, and you should found `mongo-ecs-cluster` provision with 3 Services, 3 Tasks, and 1 Container instances.

View list card 2 loaded of 2 clusters Last updated on May 14, 2021 2:04:06 PM (0m ago)

Filter in this page

Cluster name	CloudWatch monitoring...	Services	Running tasks	Pending tasks	Container instances
Default	✓ Container Insights	0	0	0	0
mongo-ecs-cluster	✓ Container Insights	3	3	0	1

mongo-ecs-primary service

Open `mongo-ecs-primary` service and check the log is starting correctly as below:-

Details Tasks Events Auto Scaling Deployments Metrics Tags **Logs**

Task status **RUNNING** STOPPED

Last updated on May 14

Filter logs × **All** 30s 5m 1h 6h 1d 1w

Timestamp (UTC+00:00)	Message
2021-05-14 13:59:53	{"t":{"date":"2021-05-14T05:59:53.278+00:00"},"s":"I", "c":"NETWORK", "id":4603701, "ctx":"LogicalSe
2021-05-14 13:59:53	{"t":{"date":"2021-05-14T05:59:53.279+00:00"},"s":"I", "c":"-", "id":4333223, "ctx":"LogicalSessionCac
2021-05-14 13:59:53	{"t":{"date":"2021-05-14T05:59:53.278+00:00"},"s":"I", "c":"REPL", "id":21392, "ctx":"ReplCoord-0", "m
2021-05-14 13:59:53	{"t":{"date":"2021-05-14T05:59:53.278+00:00"},"s":"I", "c":"REPL", "id":21393, "ctx":"ReplCoord-0", "m

EC2 instance

Explore your EC2 home console, and you should see one EC2 provisioned. Login to the EC2 using Session Manager, and run `sudo docker ps`, and you will see the current running containers, including the ECS agent container.

Instances (1) [Info](#)

Instance state: running ×

Clear filters

<input type="checkbox"/>	Name	Instance ID
<input type="checkbox"/>	mongo-ecs-EC2-Provider	i-0187dfcc15

Connect to instance

Info

Connect to your instance i-0187dfcc15f5b8c75 (mongo-ecs-EC2-Provider) using any of these options

EC2 Instance Connect

Session Manager

SSH client

EC2 Serial Console

Session Manager usage:

• Connect to your instance without SSH keys or a bastion host.

• Sessions are secured using an AWS Key Management Service key.

• You can log session commands and details in an Amazon S3 bucket or CloudWatch Logs log group.

• Configure sessions on the Session Manager Preferences page.

Cancel

Connect

Session ID: jazztong@xeri-prod-us-east-1-ec2-000000000000 Instance ID: i-0187dfcc15f5b8c75

sh-4.2\$ sudo docker ps

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
2b2fa681e5c0	bitnami/mongodb:4.4-debian-10	"/opt/bitnami/script..."	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
-primary-33-mongo-ecs-primary-ca9cc88d9b8daef87a00	bitnami/mongodb:4.4-debian-10	"/opt/bitnami/script..."	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
9c0af61dc2ac	bitnami/mongodb:4.4-debian-10	"/opt/bitnami/script..."	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
-secondary-18-mongo-ecs-secondary-9c84cdfba4d495c6bf01	bitnami/mongodb:4.4-debian-10	"/opt/bitnami/script..."	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
dc4314c4374f	bitnami/mongodb:4.4-debian-10	"/opt/bitnami/script..."	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
-arbiter-17-mongo-ecs-arbiter-9c8fb2939bdac7800900	amazon/amazon-ecs-pause:0.1.0	"/pause"	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
caf52982ceec	amazon/amazon-ecs-pause:0.1.0	"/pause"	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
-primary-33-internalecspause-daa8d290b8d99aefe501	amazon/amazon-ecs-pause:0.1.0	"/pause"	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
8fc7b83c5ab6	amazon/amazon-ecs-pause:0.1.0	"/pause"	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
-arbiter-17-internalecspause-8cb7f3fb82f5f7894e00	amazon/amazon-ecs-pause:0.1.0	"/pause"	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
b7b33a551eeb	amazon/amazon-ecs-pause:0.1.0	"/pause"	14 minutes ago	Up 14 minutes		ecs-mongo-ecs
-secondary-18-internalecspause-c89cc197cdd486f02a00	amazon/amazon-ecs-agent:latest	"/agent"	16 minutes ago	Up 16 minutes (healthy)		ecs-agent
c3d69f92bc7b						



sh-4.2\$

Network Load Balancer

Open Load balancer page, you should see one Network Load Balancer provisioned.

https://medium.com/geekculture/mongodb-replica-set-in-aws-ecs-with-terraform-4621451c6190

23/35

Filter by tags and attributes or search by keyword		
	Name	DNS name
	mongo-ecs-NLB	mongo-ecs-NLB-fa43

Test Connection from NLB

Let's test the connection to your MongoDB primary node using MongoDB Compass. Use the NLB DNS in your terraform output and fill in the credential. Create a new DB, and document to verify it is working.

If you do not change the default value, the default username is `root` and password is `mypassword`

MongoDB Compass - Connect

New Connection

☆ FAVORITE

Paste connection string

Hostname

More Options

Hostname

mongo-ecs-NLB-fa430e1

Port

27017

SRV Record



Authentication

Username / Password



Username

root

The screenshot displays the MongoDB Compass web interface. On the left sidebar, the database structure is shown: 3 DBS, 10 COLLECTIONS. The 'testdb' database is expanded, showing a collection named 'mycollection'. The main panel shows the 'Documents' tab for 'testdb.mycollection'. A filter bar is present with the filter '{ field: 'value' }'. Below the filter bar, there are buttons for 'ADD DATA', 'VIEW', and a list of documents. The documents are displayed as a list of objects, each containing an '_id' field with an ObjectId value.

Documents

testdb.mycollection

Documents Aggregations

FILTER { field: 'value' }

ADD DATA VIEW

_id: ObjectId("609e16a22e175a8eca556ebd")

_id: ObjectId("609e16a52e175a8eca556ebe")

_id: ObjectId("609e16a72e175a8eca556ebf")

_id: ObjectId("609e16a92e175a8eca556ec0")

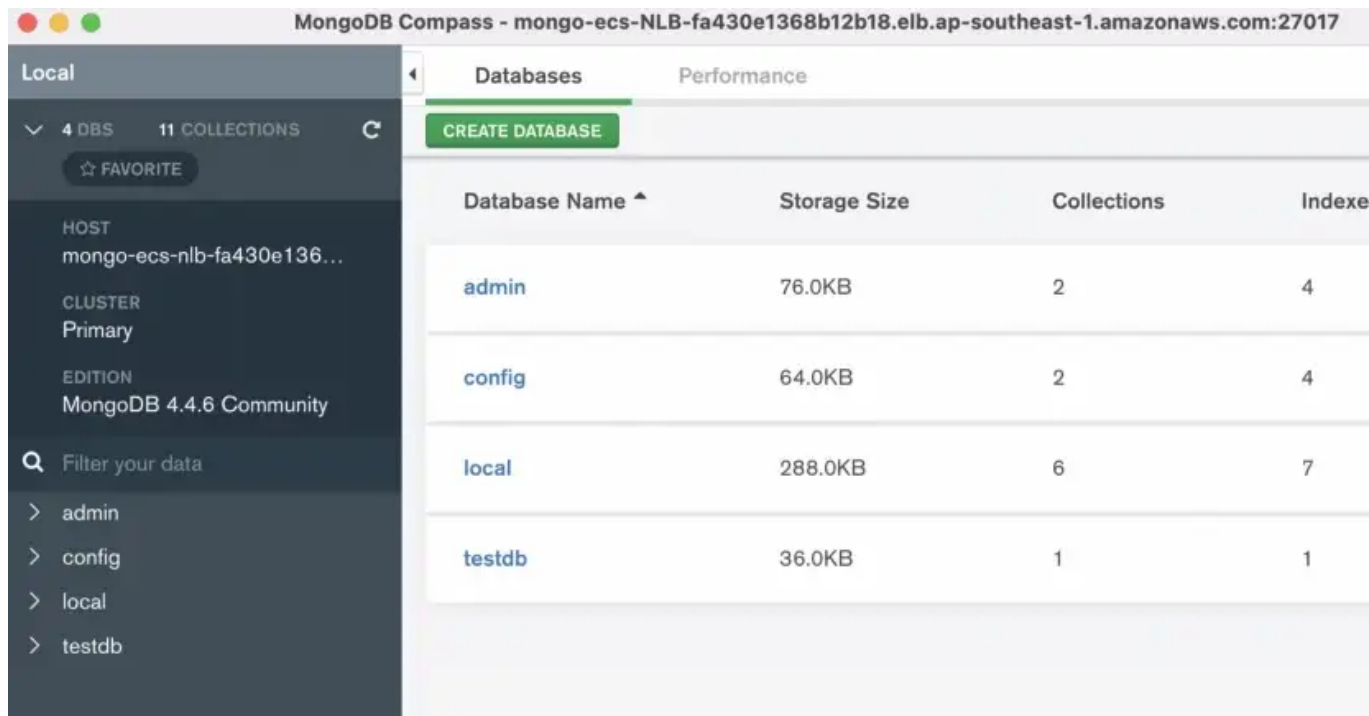
_id: ObjectId("609e16ac2e175a8eca556ec1")

Failover test

Use MongoDB Compass to connect to the primary and secondary node using NLB DNS. We will perform a failover test to bring down the primary and to ensure failover is working on the secondary node.

Connect to Primary Node

Use MongoDB Compass to connect to the primary node using NLB DNS as below.



Connect to Secondary Node

Use MongoDB Compass to connect to Secondary Node. Use the port 27018 as it configures a different port from NLB to route to the Secondary node. As it is a Secondary node, we need to specify Read Preference as `PrimaryPreferred` when we connect to it. Verify that the Secondary Node is not allowed to perform write action.

Hostname

More Options

Hostname

mongo-ecs-NLB-fa430e1


Port

27018

SRV Record

☐

Authentication

Username / Password 

Username

root

Password

.....

New Connection

☆ FAVORITE

Paste connection string

Hostname

More Options

Replica Set
Name

Read Preference

Primary Preferred



SSL

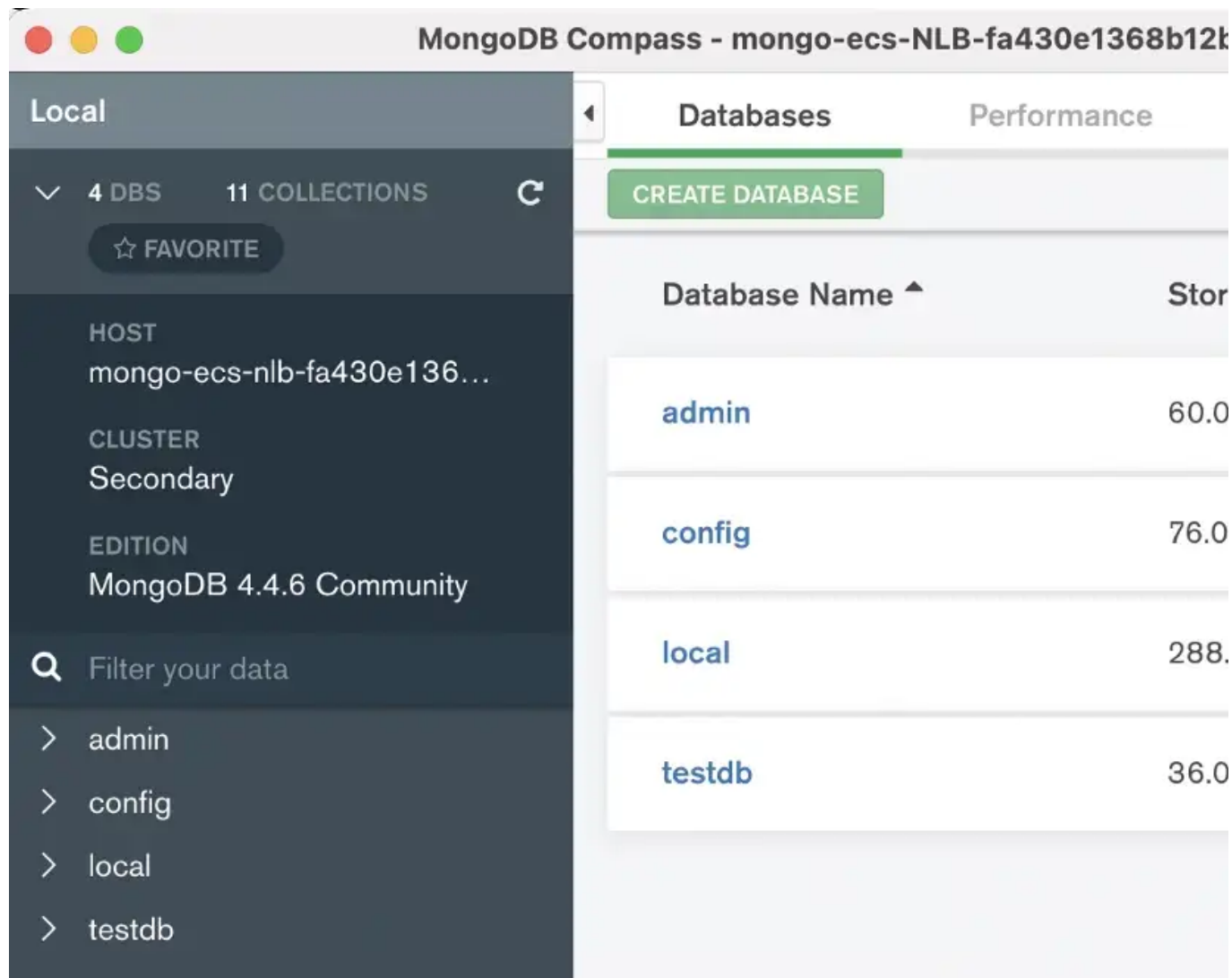
None



SSH Tunnel

None





Stop MongoDB primary task

Open ECS home, and look up the MongoDB Primary service. In the task list, stop the task to simulate the downtime of MongoDB primary node.

[Clusters](#) > [mongo-ecs-cluster](#) > Task: 763347e93a1947a8af633338ea564e9d

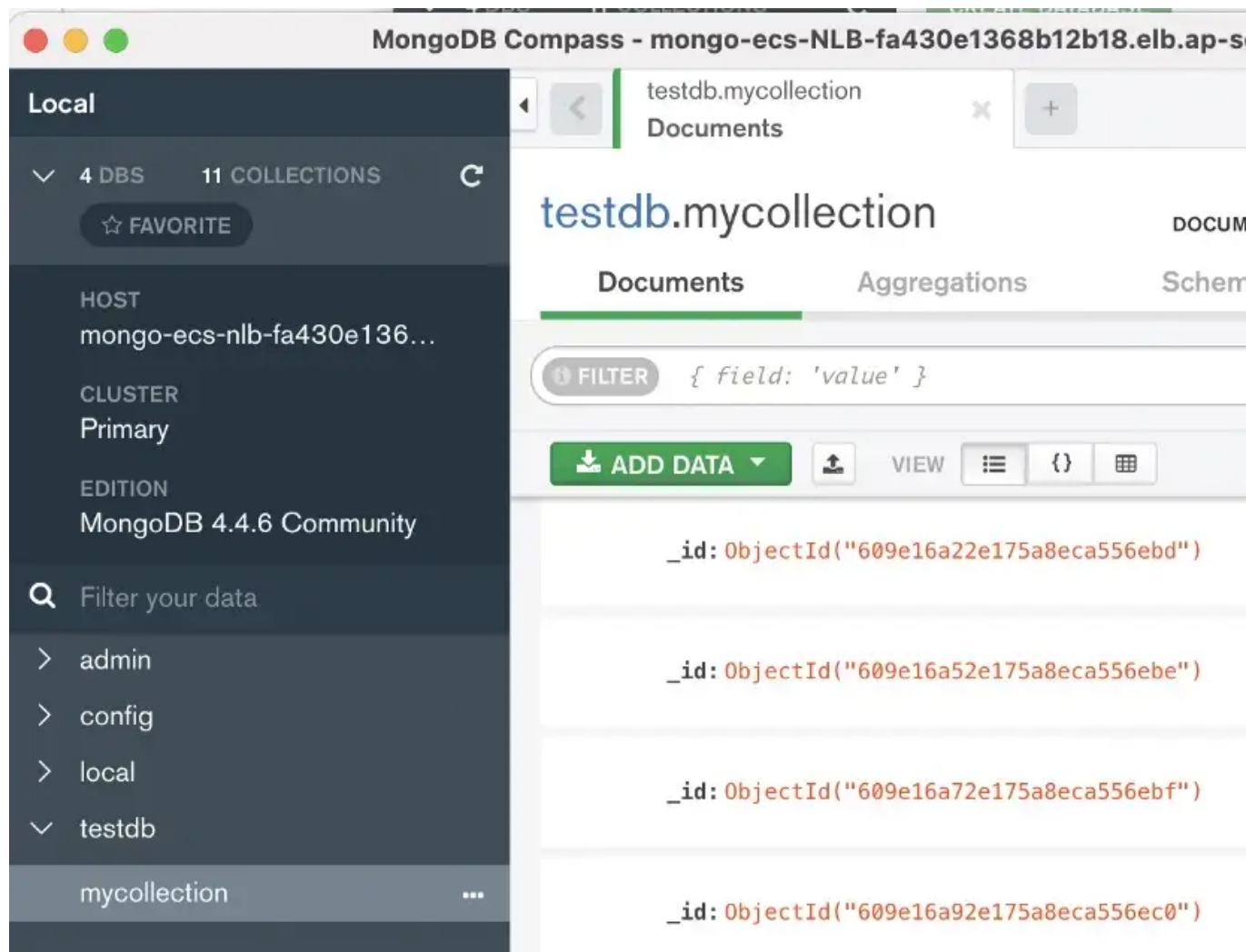
Task : 763347e93a1947a8af633338ea564e9d

[Run more like this](#)[Stop](#)[Details](#)[Tags](#)[Logs](#)

Observe Secondary Node become Primary

Observe the Secondary node, the failover will happen within 1 min time.

Secondary node will become primary and allow to perform write action.



Old Primary Node recovering

One of the Bitnami image features is to recover the replica set to force the original Primary node to become Primary. Observe the MongoDB Primary ECS service until it creates a new task, and it will take over the primary node

when it is stable. It allows the system to go back to its original state without a change on the application side.

Take away

We use Terraform to build up all the components, and it allows us to easily rebuild if something goes wrong. I hope you enjoy this solution, and let me know if there is any issue to use the template.

jazztong/mongo-ecs-ha-tf

Demonstrate Mongo ECS Replica setup with terraform, it will setup 3 MongoDB nodes in ECS with EC2 launch type Run...

[github.com](#)

Mongodb

Terraform

Ecs

High Availability

Mongodb Replica

Sign up for Geek Culture Hits

By Geek Culture

Subscribe to receive top 10 most read stories of Geek Culture — delivered straight into your inbox, once a week. [Take a look.](#)

Your email



Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

[About](#) [Help](#) [Terms](#) [Privacy](#)