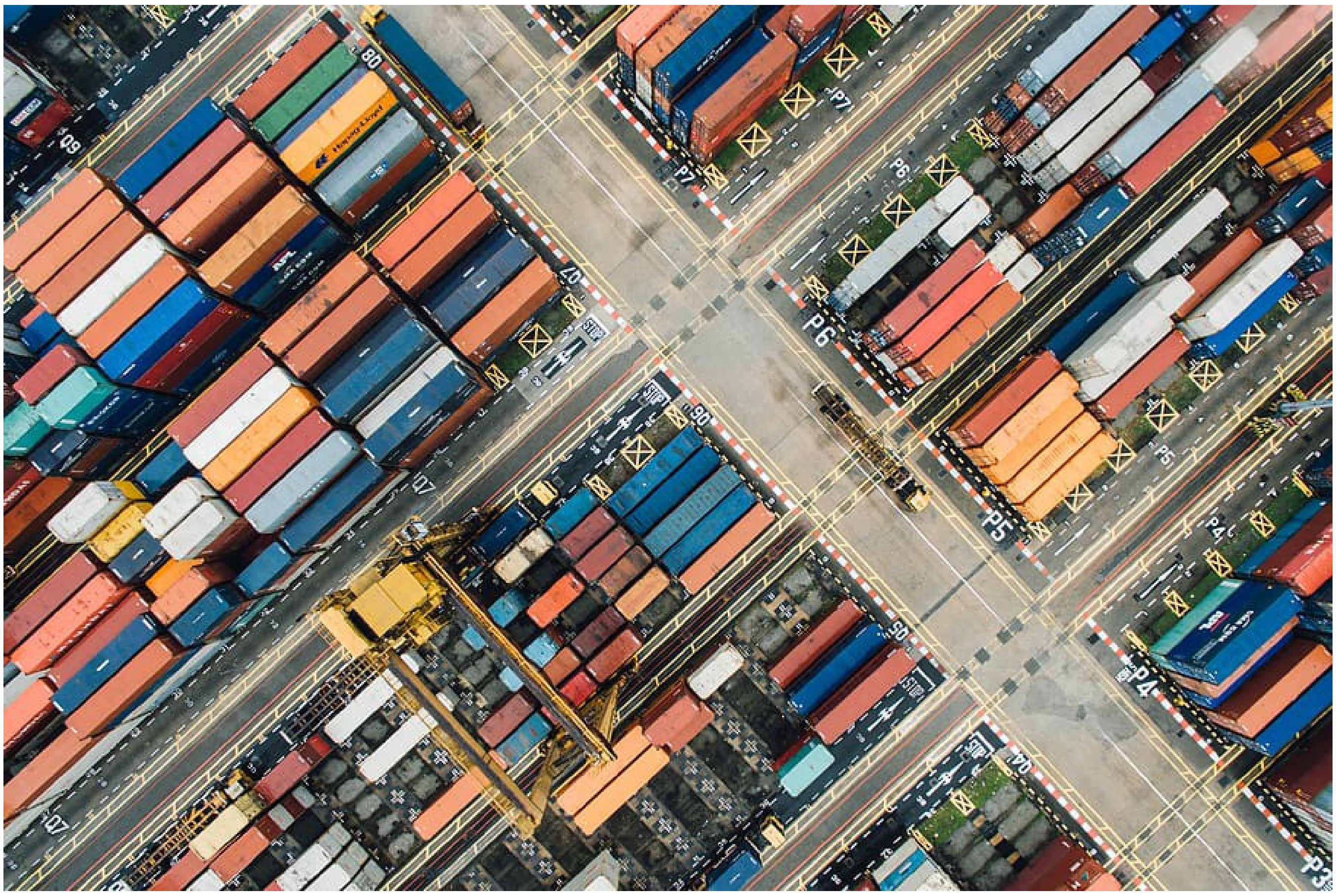
How To: Run A Cost Effective Blue Green Like Deployment Using ECS Or Fargate

Craig Godden-Payne
Mar 9 · 5 min read ★



ECS and Fargate are serverless solutions provided by amazon, which can run pretty much anything that you can run in a docker container.

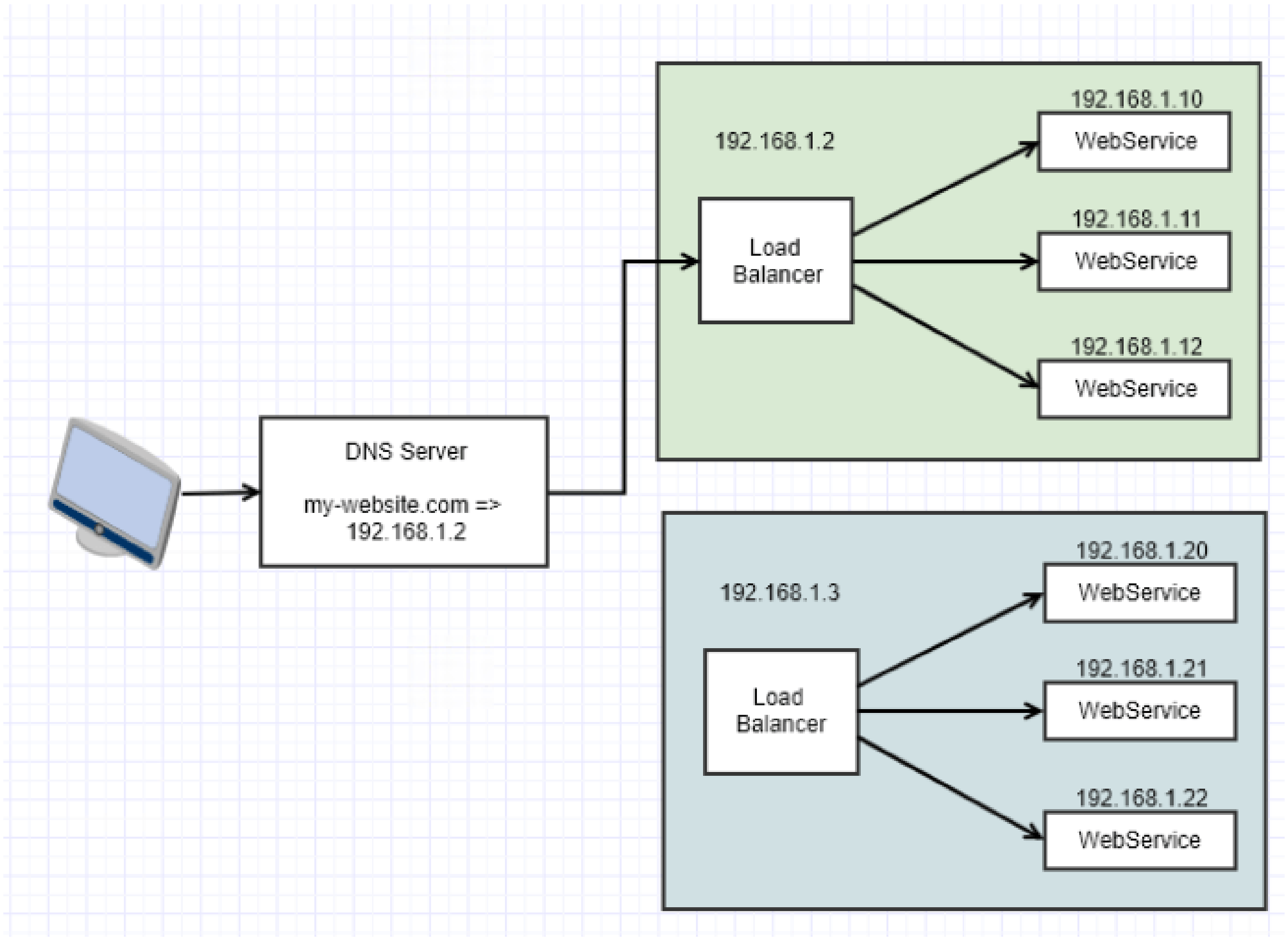
There is a feature that isn't talked about, but that is the ability to perform a blue green like deployment in a cost effective way.

I mention blue green like, as it is not the traditional blue green deployment.

What is a blue green deployment?

Traditionally, a blue-green deployment was described as a technique that reduces downtime and risk by running two identical production environments called Blue and Green.

At any time, only one of the environments is live, with the live environment serving all production traffic. Because there is already another environment *ready to go,* it's possible to switch environments if something happens to the green environment or a new version needs to be released, and can be really handy in a disaster recovery scenario.



What is the cost effective blue green like deployment using ECS or Fargate?

Since your docker containers are versioned within ECR, essentially it is possible to spin up the previous version of the container, and roll the deployment using a load balancer when you want to deploy a new version or roll back.

This is all handled by ECS or Fargate services, and is automatic.

But how does this compare?

There are pros and cons for using this technique, here is a quick summary:

Pros

The cost is reduced, as you are only running multiple environments for a short period of time

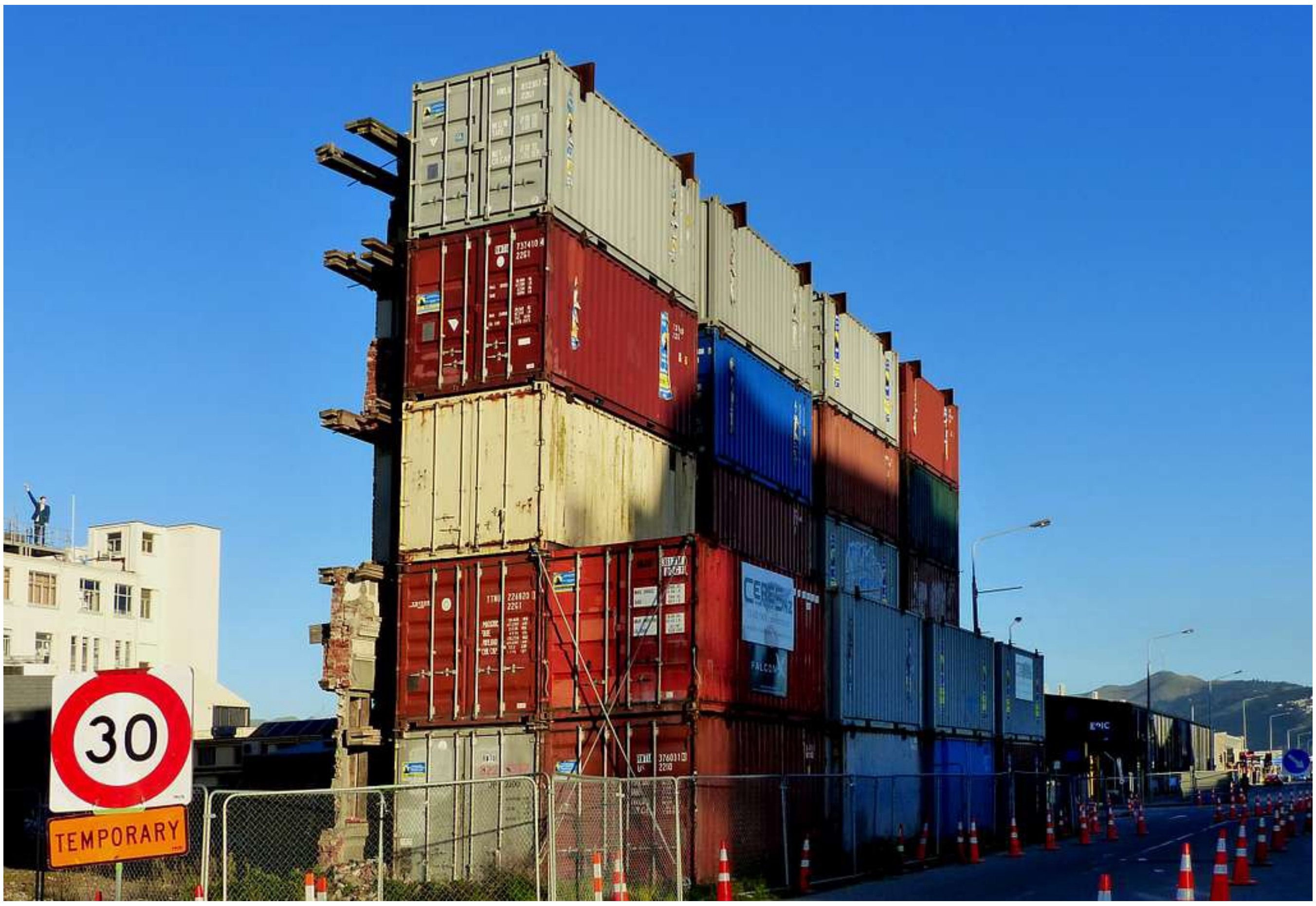
The rolling deployment is handled for you, and it will never switch to the new environment if it is failing

The rolling deployment is rather fast, usually within a couple of minutes

Cons

If the deployment fails, it is not as obvious, usually you see in the console that the service is stuck in a draining state

Fargate does not utilise lifecycle hooks, so if you have a task which needs to be drained, or long running, it can cause issues.



How do you configure this?

So you know know how this compares, we should have a look at how this works in a real life scenario.

You can handle the configuration of the services using terraform, here are some examples.

```
BASE_REPO=XXXXXXXXXXXX.dkr.ecr.eu-west-2.amazonaws.com
IMAGE_NAME=${dir:2}
VERSION_LATEST=latest
VERSION=2.0
echo "ImageName:"$IMAGE_NAME
eval $(aws ecr get-login --region eu-west-2 --no-include-email)
sleep 1

docker build $dir -t $IMAGE_NAME:$VERSION
docker tag $IMAGE_NAME:$VERSION
$BASE_REPO/$IMAGE_NAME:$VERSION_LATEST
docker tag $IMAGE_NAME:$VERSION $BASE_REPO/$IMAGE_NAME:$VERSION
docker push $BASE_REPO/$IMAGE_NAME:$VERSION
docker push $BASE_REPO/$IMAGE_NAME:$VERSION_LATEST
```

Example build script for building the docker container and pushing to ECR

```
resource "aws_ecs_service" "ecs-service-with-loadbalancer" {
  name                              = "${var.app_name}"
  cluster                           = "${data.aws_ecs_cluster.app-
container-host.id}"
  task_definition                   =
"${aws_ecs_task_definition.definition.arn}"
  scheduling_strategy               = "REPLICA"
  desired_count                     = "${var.desired_count}"
  health_check_grace_period_seconds = "${var.health_check_period}"
  iam_role                          = "${aws_iam_role.api.name}"

  ordered_placement_strategy {
    type  = "spread"
    field = "host"
  }

  load_balancer {
    container_name   = "${var.app_name}"
    container_port   = "${var.container_port}"
    target_group_arn = "${aws_lb_target_group.api.arn}"
  }
}
```

Example ECS service configuration

```
resource "aws_iam_role" "api" {
  name = "${var.app_name}-role"

  assume_role_policy = <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": "sts:AssumeRole",
      "Principal": {
        "Service": [
          "ecs.amazonaws.com",
          "ecs-tasks.amazonaws.com"
        ]
      },
      "Effect": "Allow",
      "Sid": ""
    }
  ]
}
EOF
}
```

Iam role configuration

```
resource "aws_lb_target_group" "api" {
  protocol   = "HTTP"
  vpc_id     = "${data.aws_vpc.vpc.id}"
  name       = "${var.app_name}"
  slow_start = 0
}

resource "aws_lb_listener_rule" "api" {
  listener_arn = "${data.aws_alb_listener.public.arn}"
  priority     = "${var.lb_rule_number}"

  action {
    type             = "forward"
    target_group_arn = "${aws_lb_target_group.api.arn}"
  }

  condition {
    field  = "path-pattern"
    values = ["${var.lb_pattern}"]
  }
}
```

Example of setup for load balancer

```
resource "aws_ecs_task_definition" "definition" {
  family             = "${var.app_name}"
  network_mode       = "bridge"
  task_role_arn      = "${aws_iam_role.api.arn}"
  execution_role_arn = "${aws_iam_role.api.arn}"

  container_definitions = <<DEFINITION
[
  {
    "name": "${var.app_name}",
    "image":
"${data.aws_caller_identity.current.account_id}.dkr.ecr.eu-west-
2.amazonaws.com/${var.app_name}:${var.application_version}",
    "essential": true,
    "privileged": true,
    "memoryReservation": ${var.task_memory},
    "portMappings": [
      {
        "containerPort": ${var.container_port},
        "protocol": "tcp"
      }
    ],
    "environment": [
      {
          "name": "ApplicationVersion",
          "value": "${var.application_version}"
      }
    ],
    "requiresAttributes": [
        {
        "value": null,
        "name": "com.amazonaws.ecs.capability.ecr-auth",
        "targetId": null,
        "targetType": null
        },
        {
        "value": null,
        "name": "com.amazonaws.ecs.capability.task-iam-role",
        "targetId": null,
        "targetType": null
        },
        {
        "value": null,
        "name": "com.amazonaws.ecs.capability.docker-remote-
api.1.19",
        "targetId": null,
        "targetType": null
        }
    ]
  }
]
DEFINITION
}
```

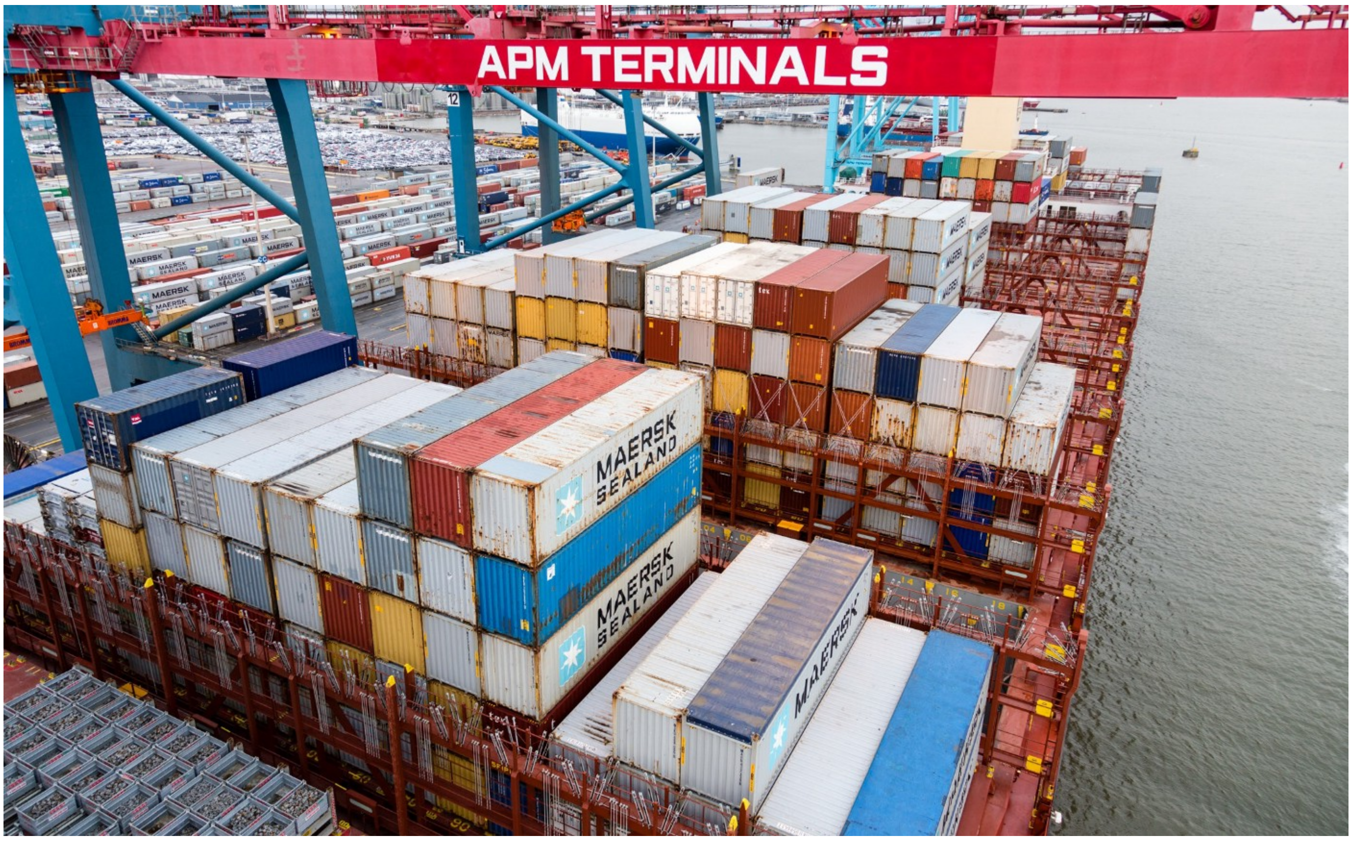
Example of task definition configuration

What is the update script we use?

There are multiple ways to trigger the deployment, but the one I prefer is to call aws using a cli command to trigger. You can update the task definition which will do the same kind of thing, but using the cli has much better control.

```
aws ecs update-service \
--region eu-west-2 \
--cluster myservice \
--service myservice-myapplication \
--task-definition myservice-myapplication-task \
--force-new-deployment
```

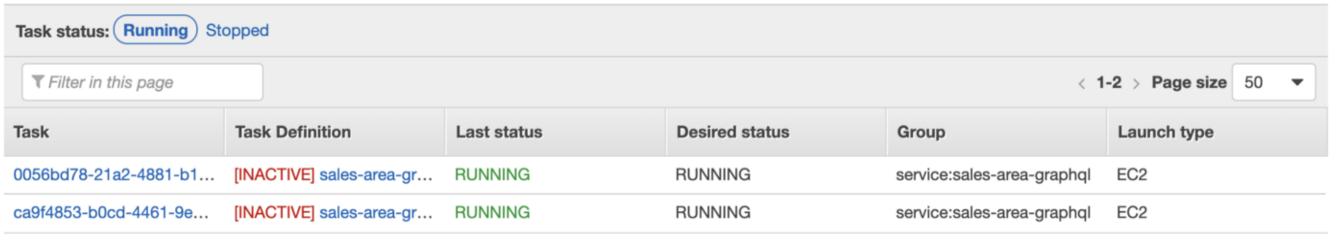
The important part of the script being the force-new-deployment, this will trigger a deployment even if there are no changes



And what does this look like in a real life scenario?

After the update service script has been run, you will see the following in AWS.

1. Tasks are marked as inactive in AWS and starts the draining process

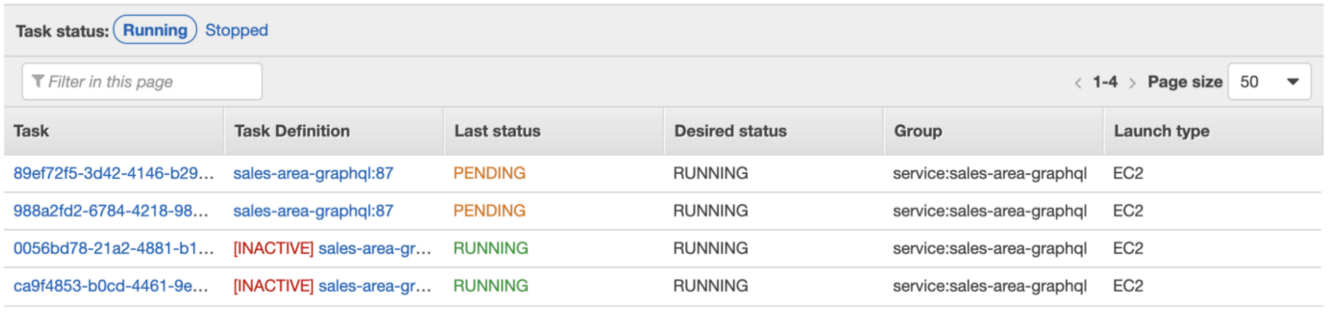


Task status: Running Stopped

Filter in this page — 1-2 — Page size 50

Task	Task Definition	Last status	Desired status	Group	Launch type
0056bd78-21a2-4881-b1...	[INACTIVE] sales-area-gr...	RUNNING	RUNNING	service:sales-area-graphql	EC2
ca9f4853-b0cd-4461-9e...	[INACTIVE] sales-area-gr...	RUNNING	RUNNING	service:sales-area-graphql	EC2

2. The new task is started, and enters a pending state

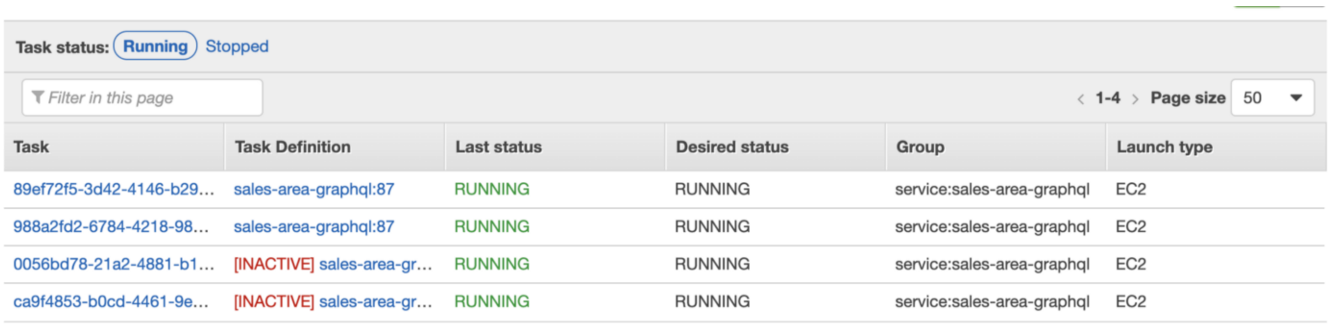


Task status: Running Stopped

Filter in this page — 1-4 — Page size 50

Task	Task Definition	Last status	Desired status	Group	Launch type
89ef72f5-3d42-4146-b29...	sales-area-graphql:87	PENDING	RUNNING	service:sales-area-graphql	EC2
988a2fd2-6784-4218-98...	sales-area-graphql:87	PENDING	RUNNING	service:sales-area-graphql	EC2
0056bd78-21a2-4881-b1...	[INACTIVE] sales-area-gr...	RUNNING	RUNNING	service:sales-area-graphql	EC2
ca9f4853-b0cd-4461-9e...	[INACTIVE] sales-area-gr...	RUNNING	RUNNING	service:sales-area-graphql	EC2

3. The new task starts successfully, and a health check is performed from the load balancer to container:port

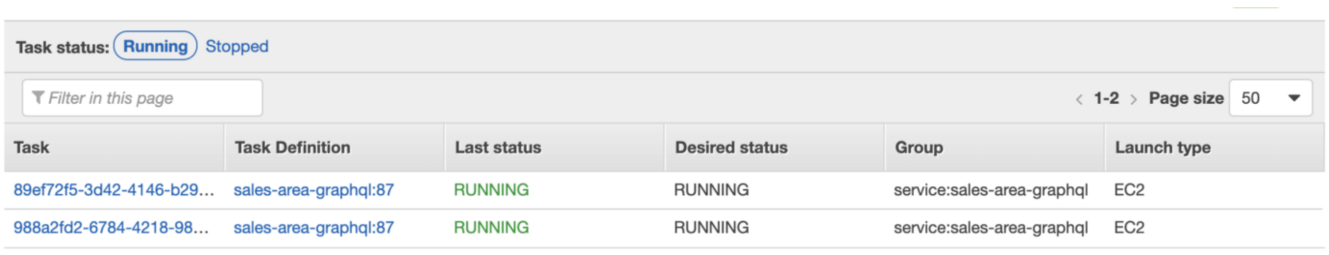


Task status: Running Stopped

Filter in this page ‹ 1-4 › Page size 50

Task	Task Definition	Last status	Desired status	Group	Launch type
89ef72f5-3d42-4146-b29...	sales-area-graphql:87	RUNNING	RUNNING	service:sales-area-graphql	EC2
988a2fd2-6784-4218-98...	sales-area-graphql:87	RUNNING	RUNNING	service:sales-area-graphql	EC2
0056bd78-21a2-4881-b1...	[INACTIVE] sales-area-gr...	RUNNING	RUNNING	service:sales-area-graphql	EC2
ca9f4853-b0cd-4461-9e...	[INACTIVE] sales-area-gr...	RUNNING	RUNNING	service:sales-area-graphql	EC2

4. After 30 seconds or so, the old version of the containers drained from the cluster

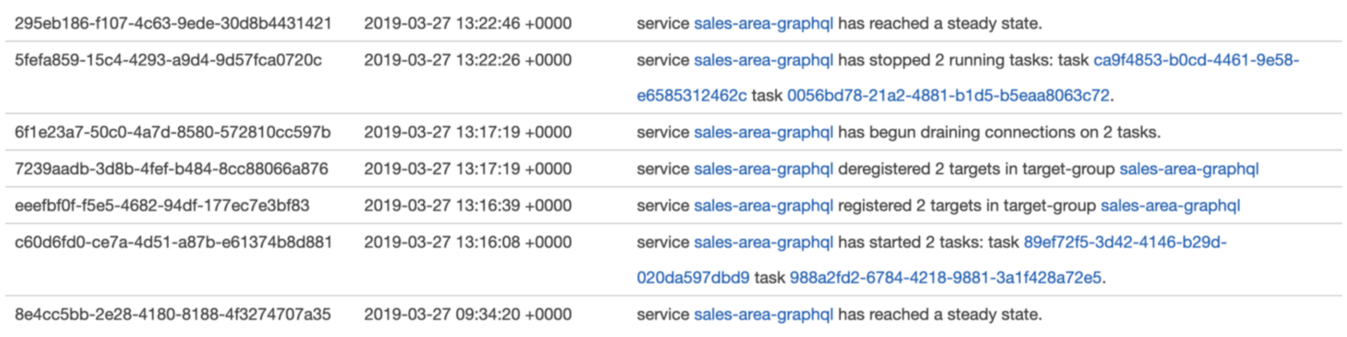


Task status: Running Stopped

Filter in this page ‹ 1-2 › Page size 50

Task	Task Definition	Last status	Desired status	Group	Launch type
89ef72f5-3d42-4146-b29...	sales-area-graphql:87	RUNNING	RUNNING	service:sales-area-graphql	EC2
988a2fd2-6784-4218-98...	sales-area-graphql:87	RUNNING	RUNNING	service:sales-area-graphql	EC2

Here is an example of the log, which shows the blue green deployment.



```
295eb186-f107-4c63-9ede-30d8b4431421  2019-03-27 13:22:46 +0000  service sales-area-graphql has reached a steady state.
5fefa859-15c4-4293-a9d4-9d57fca0720c  2019-03-27 13:22:26 +0000  service sales-area-graphql has stopped 2 running tasks: task ca9f4853-b0cd-4461-9e58-
                                                                 e6585312462c task 0056bd78-21a2-4881-b1d5-b5eaa8063c72.
6f1e23a7-50c0-4a7d-8580-572810cc597b  2019-03-27 13:17:19 +0000  service sales-area-graphql has begun draining connections on 2 tasks.
7239aadb-3d8b-4fef-b484-8cc88066a876  2019-03-27 13:17:19 +0000  service sales-area-graphql deregistered 2 targets in target-group sales-area-graphql
eeefbf0f-f5e5-4682-94df-177ec7e3bf83  2019-03-27 13:16:39 +0000  service sales-area-graphql registered 2 targets in target-group sales-area-graphql
c60d6fd0-ce7a-4d51-a87b-e61374b8d881  2019-03-27 13:16:08 +0000  service sales-area-graphql has started 2 tasks: task 89ef72f5-3d42-4146-b29d-
                                                                 020da597dbd9 task 988a2fd2-6784-4218-9881-3a1f428a72e5.
8e4cc5bb-2e28-4180-8188-4f3274707a35  2019-03-27 09:34:20 +0000  service sales-area-graphql has reached a steady state.
```

Blue Green Deployment   Deployment   DevOps   Serverless Architecture   Cost Savings