



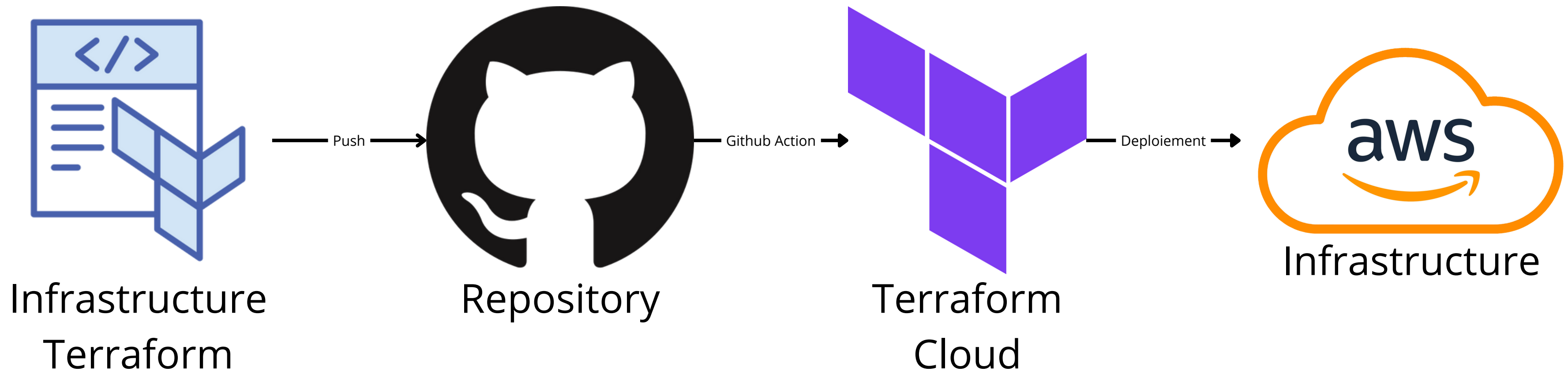
**5 millions de messages
Discord envoyés chaque
seconde.**

Une présentation de Thomas, Dylan et Sylvain

Environnement de travail



Déploiement automatique



Github Action

Lors d'un push sur la branche main

```
name: "Terraform"

on:
  push:
    branches:
      - main
```

Github Action

Récupération du code & Connexion a Terraform Cloud

```
- name: Checkout
  uses: actions/checkout@v3

- name: Setup Terraform
  uses: hashicorp/setup-terraform@v1
  with:
    cli_config_credentials_token: ${ secrets.TF_API_TOKEN }
```

Github Action

Déploiement de l'infrastructure

```
- name: Terraform Apply
  if: github.ref == 'refs/heads/main' && github.event_name == 'push'
  run: terraform apply -auto-approve -input=false
  working-directory: ${ env.TERRAFORM_DIR }
```

Terraform Cloud

Déploiement de l'infrastructure

Triggered via CLI


✓ Planned and finished

Plan duration

Less than a minute

Resources to be changed

+2 ~1 -0

 API Integration triggered a **speculative plan** from CLI 2 months ago Run Details

✓ **Plan finished** 2 months ago



Resources: 2 to add, 1 to change, 0 to destroy



Started 2 months ago > Finished 2 months ago



+ 2 to create

~ 1 to change

Filter by action ☐ Show data sources Terraform 1.10.3 Download raw log

> +  aws_elasticache_cluster.example 

> ~  aws_instance.ecs_instance 

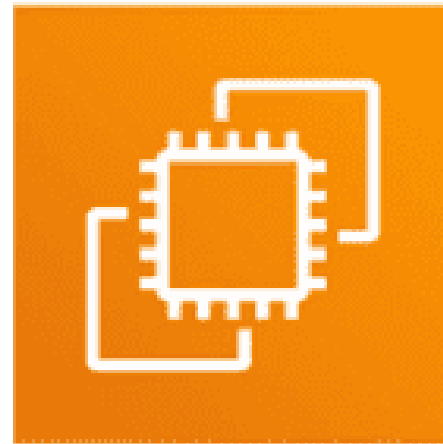
> +  aws_security_group.elasticache_sg 

> **Outputs** 1 planned to change

Download Sentinel mocks ⓘ Sentinel mocks can be used for [testing your Sentinel policies](#)

1ère itération

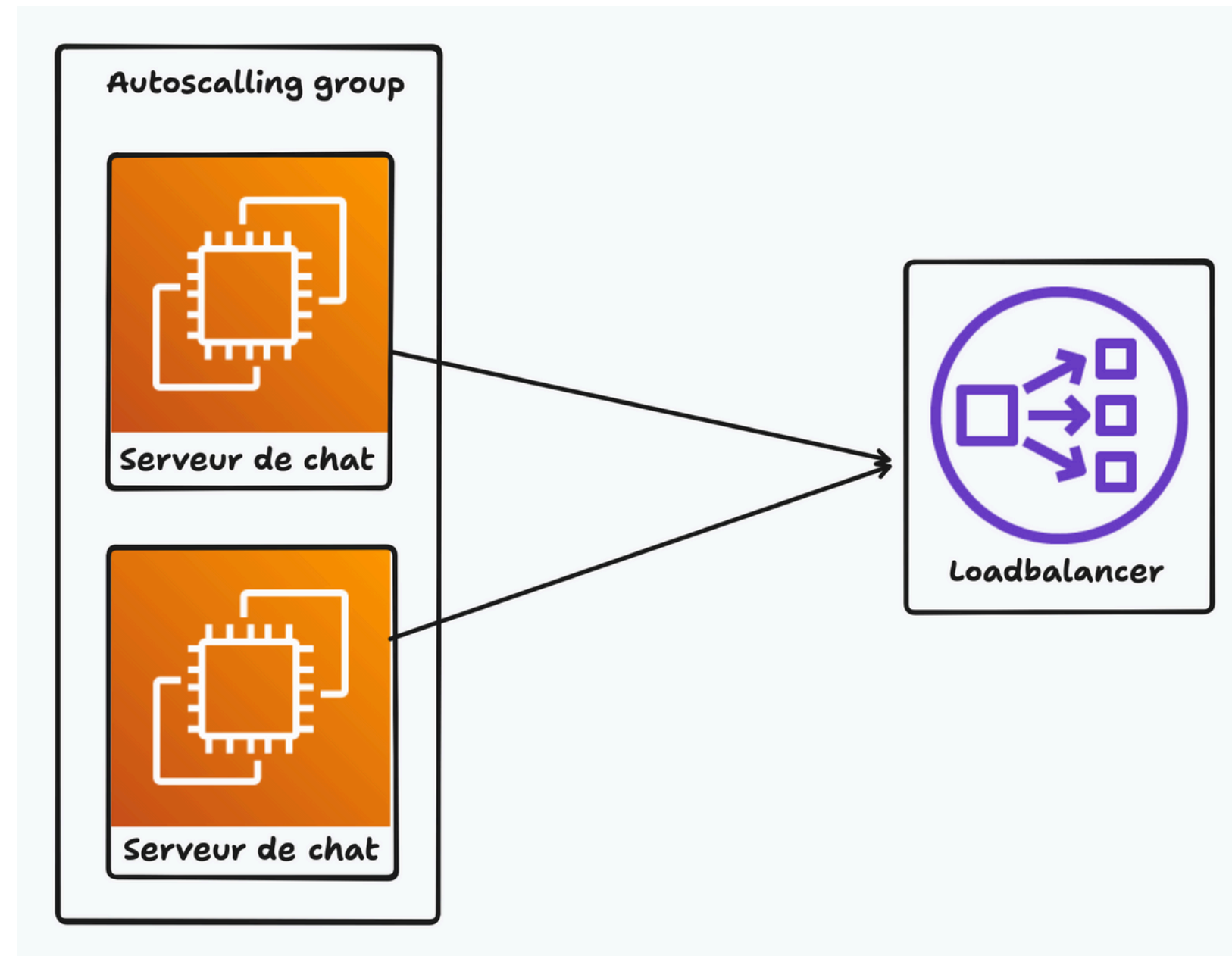
Systeme simple



Amazon
EC2

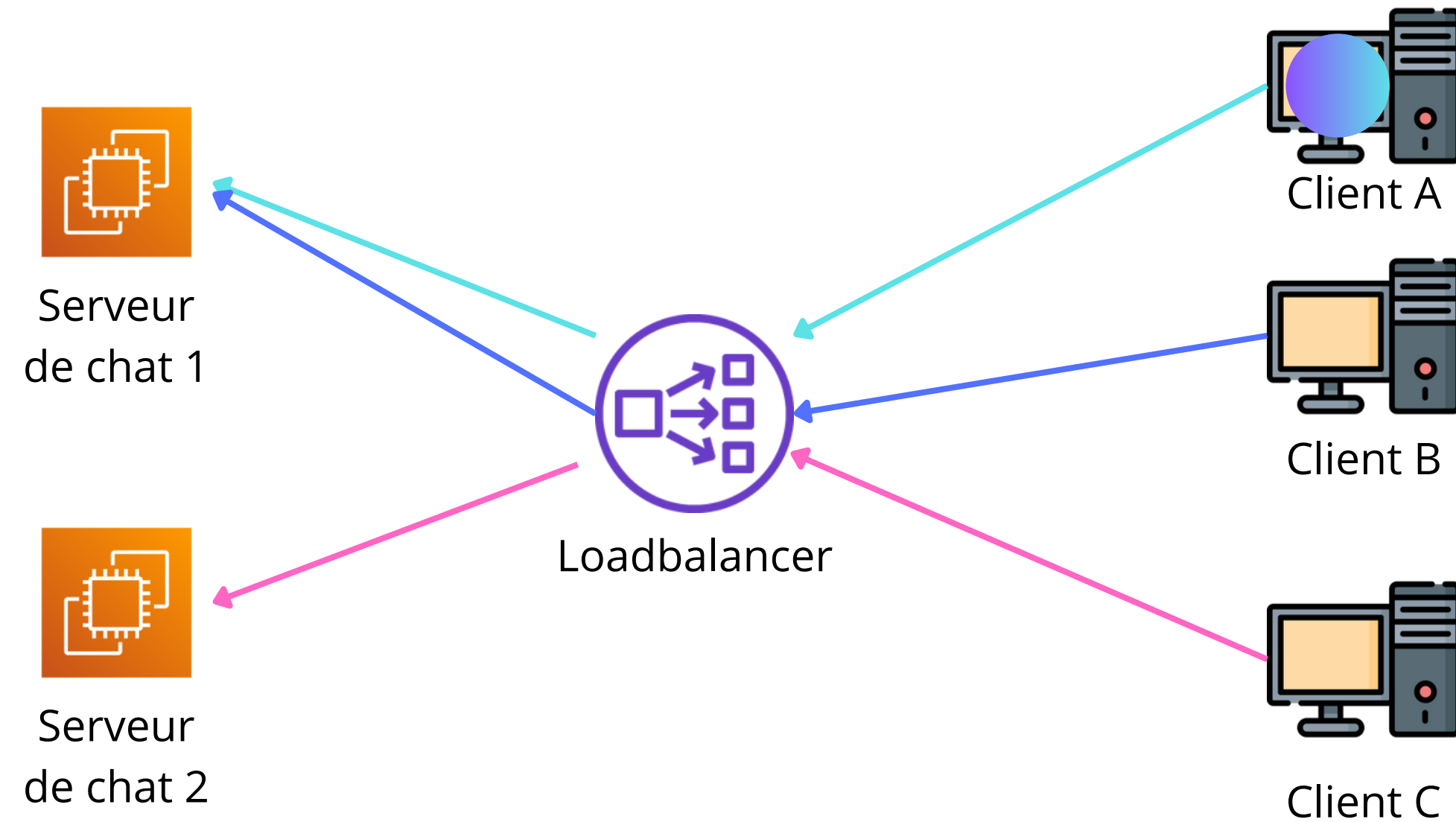
2ème itération

Autoscaling + Loadbalancer



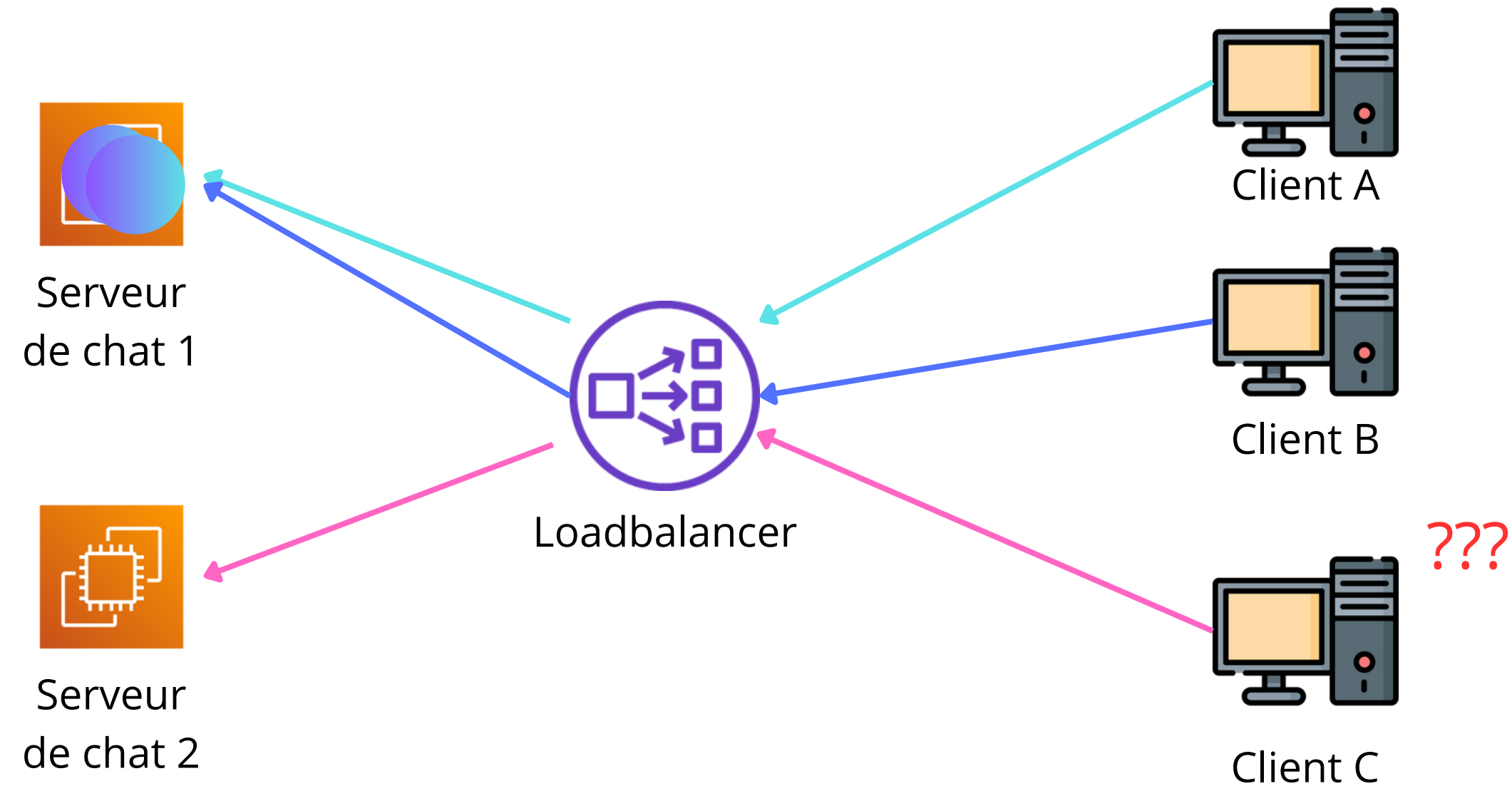
2ème itération

Problématique



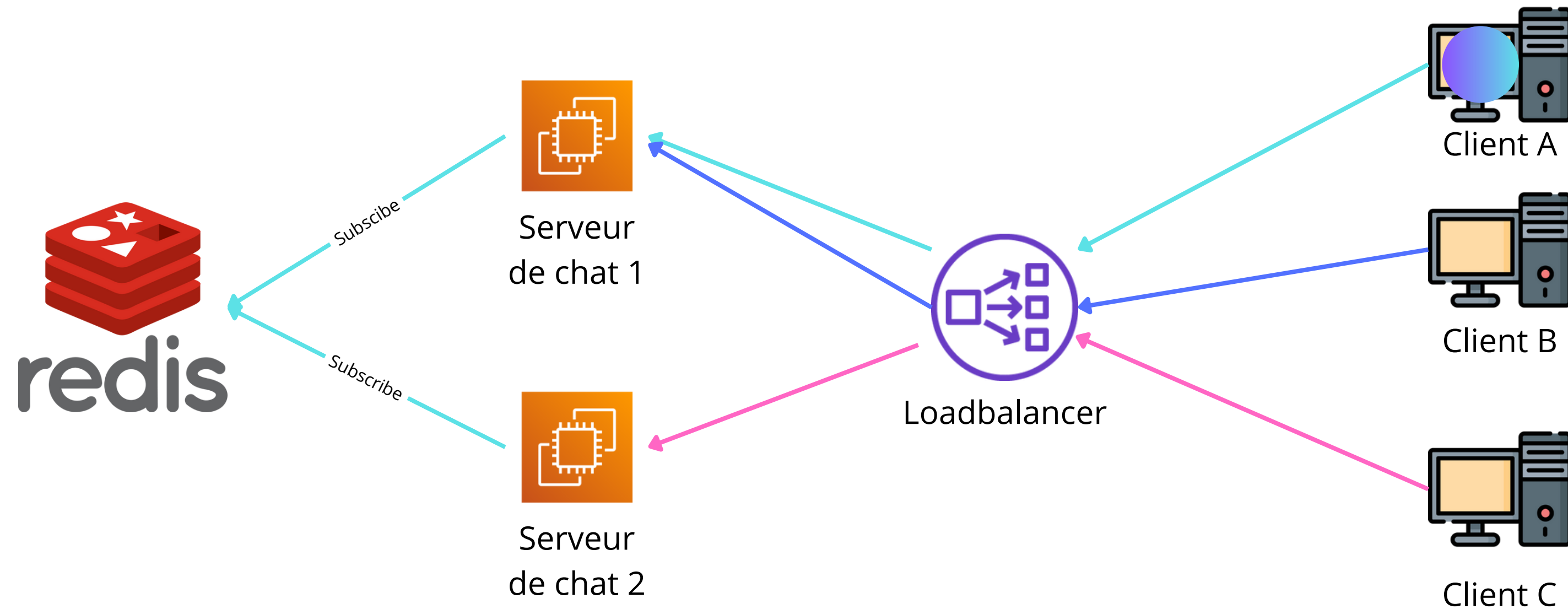
2ème itération

Problématique



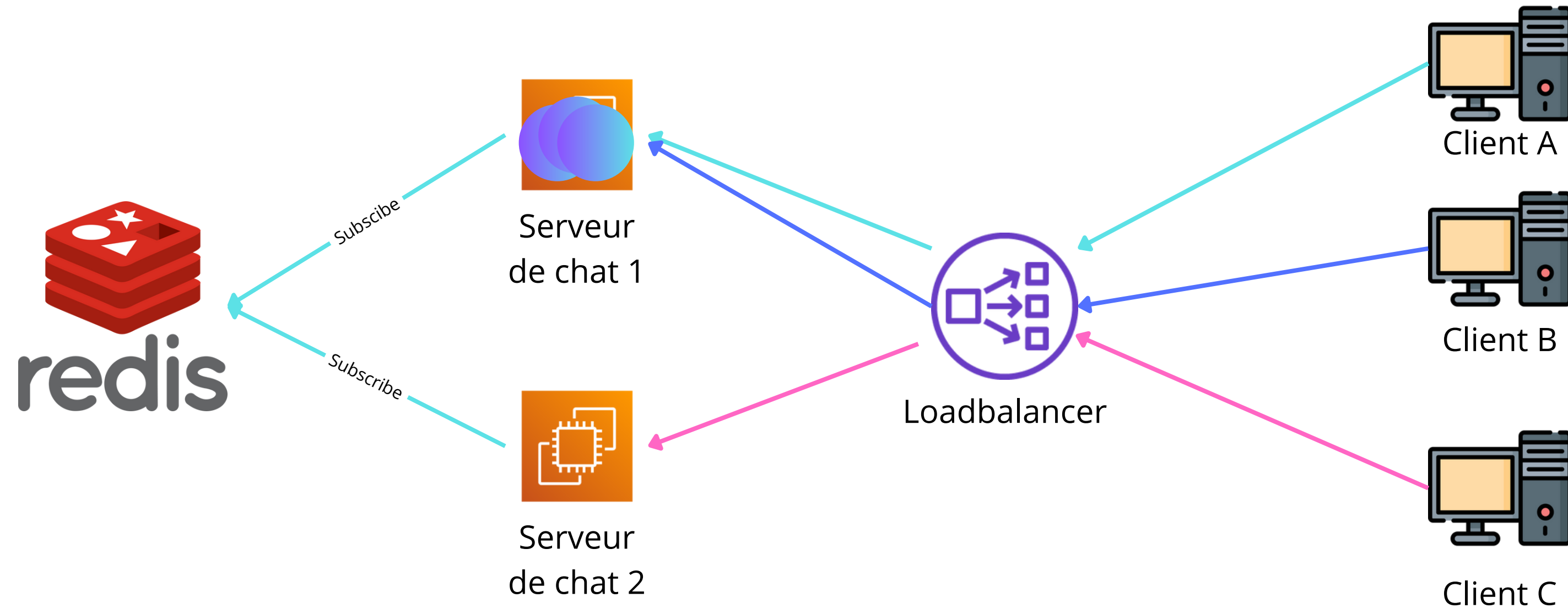
2ème itération

Problématique



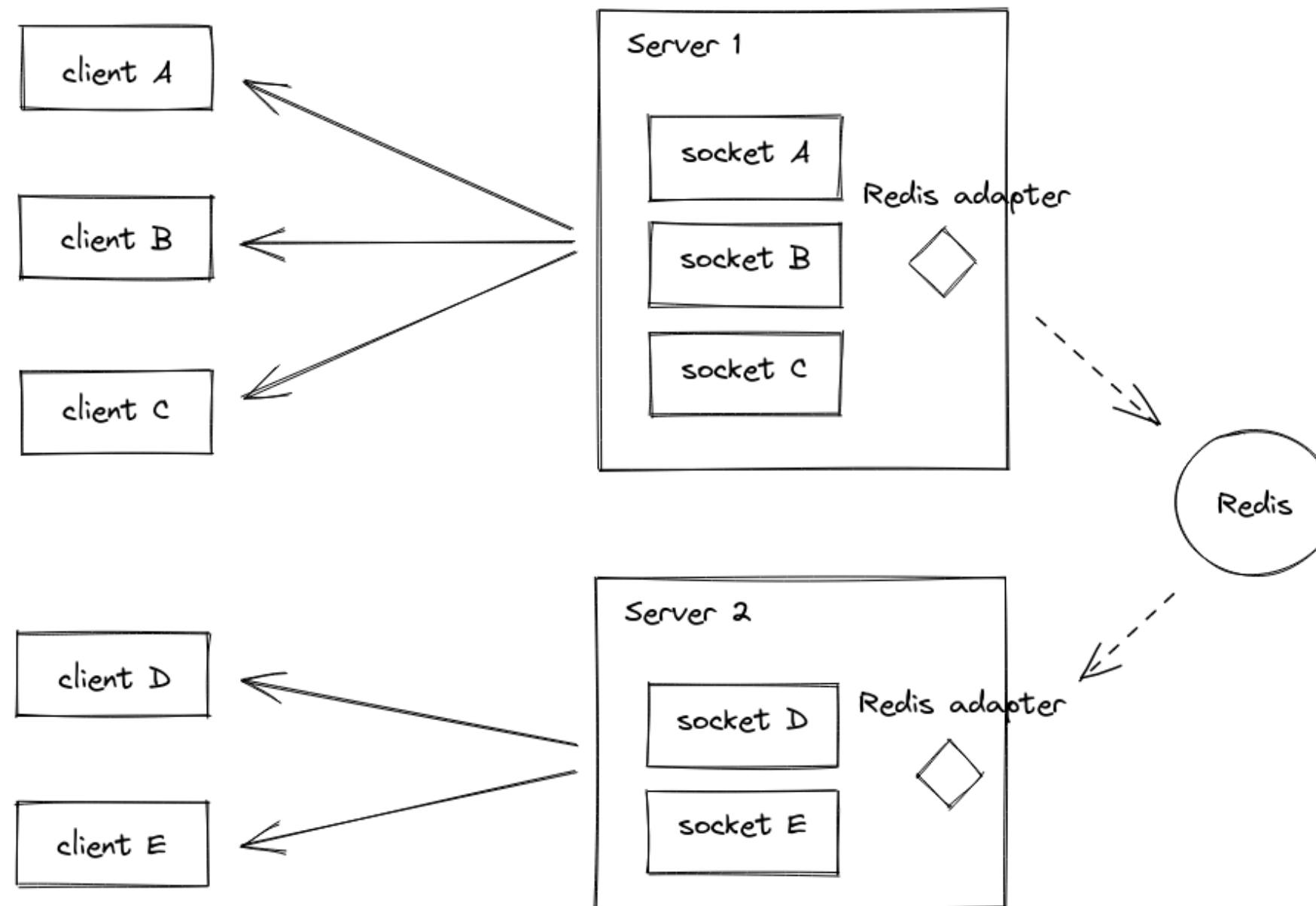
2ème itération

Problématique



2ème itération

Solution : Redis Pub Sub



3ème itération

Elastic Container Service



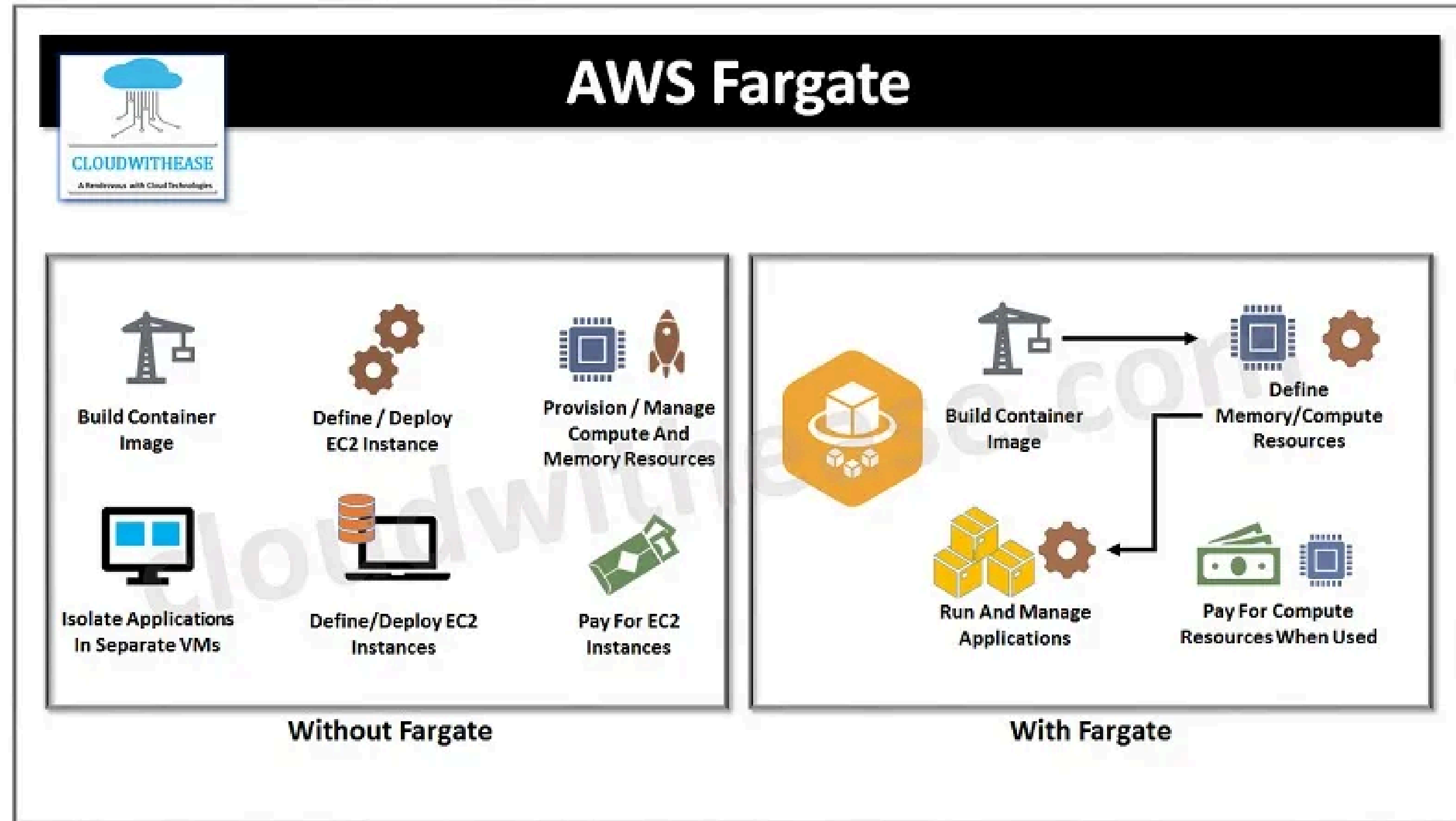
Comment fonctionne ECS ?

3ème itération

ECS

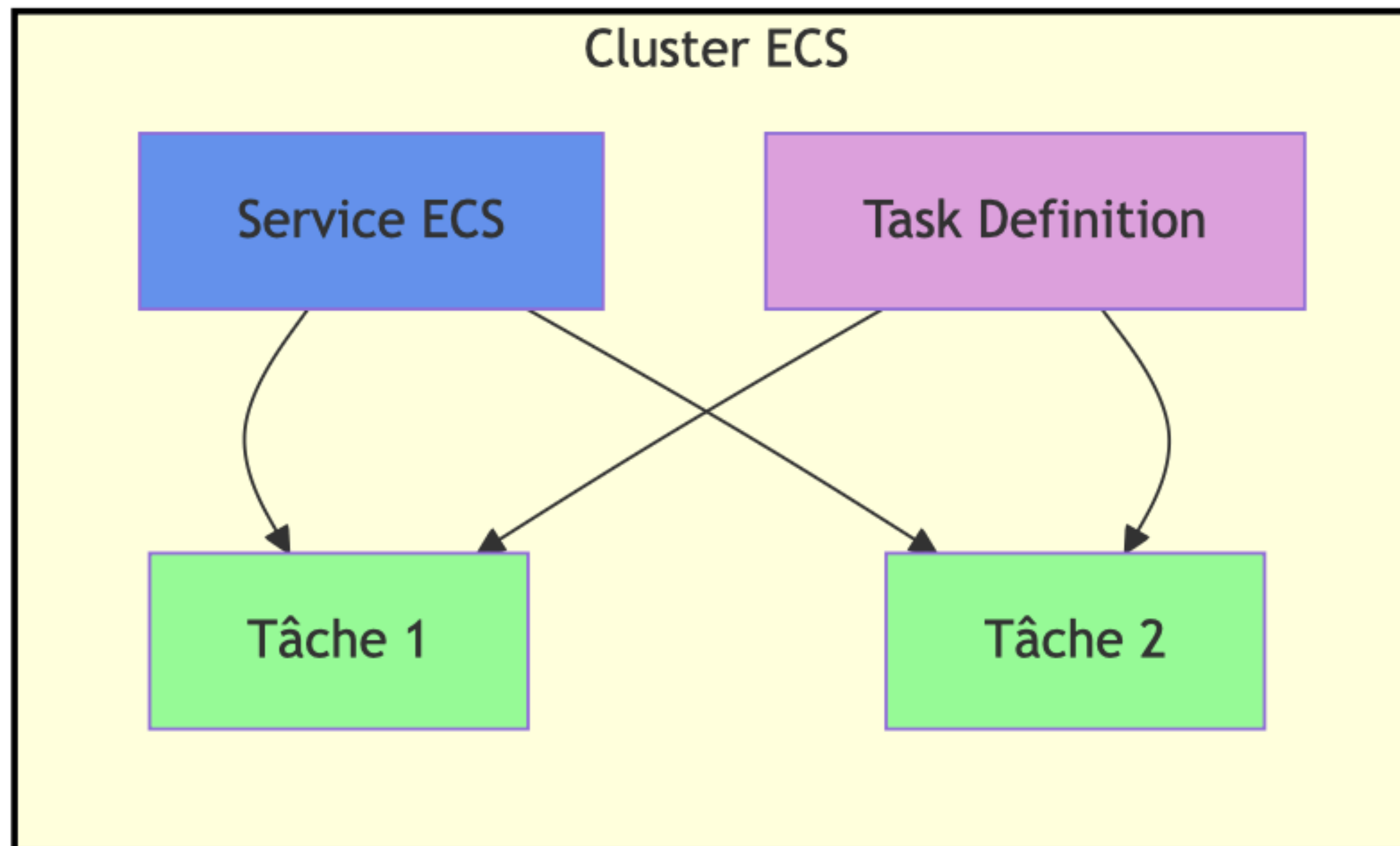
- Orchestration de conteneurs
- Gestion simplifiée de l'infrastructure
- Scalabilité
- Sécurité
- Déploiement via des Task Definitions
- Intégration CI/CD

3ème itération



3ème itération

ECS



Task definition

```
resource "aws_ecs_task_definition" "std-ecs-task" {
  family                = "std-ecs-task"
  requires_compatibilities = ["FARGATE"]
  network_mode          = "awsvpc"
  cpu                    = 1024
  memory                 = 2048

  task_role_arn      = aws_iam_role.ecs_task_role_chat.arn
  execution_role_arn = aws_iam_role.ecs_execution_role_chat.arn

  container_definitions = jsonencode([
    {
      name       = "std-ecs-chat"
      image      = "ghcr.io/thfx31/std/chat-server:latest"
      cpu        = 1024
      memory     = 2048
      portMappings = [
        {
          containerPort = 3000
          hostPort       = 3000
        }
      ]

      environment = [
        {
          name  = "ELASTICACHE_ENDPOINT"
          value = var.elasticache_endpoint
        }
      ]
    }
  ])
}
```

3ème itération

Démo du déploiement

- Ajout d'une instance
- Changement du background

ECS Service

```
resource "aws_ecs_service" "std-ecs-service" {
  name           = "std-ecs-service"
  cluster        = aws_ecs_cluster.std-ecs-cluster.id
  task_definition = aws_ecs_task_definition.std-ecs-task.arn
  desired_count  = 2

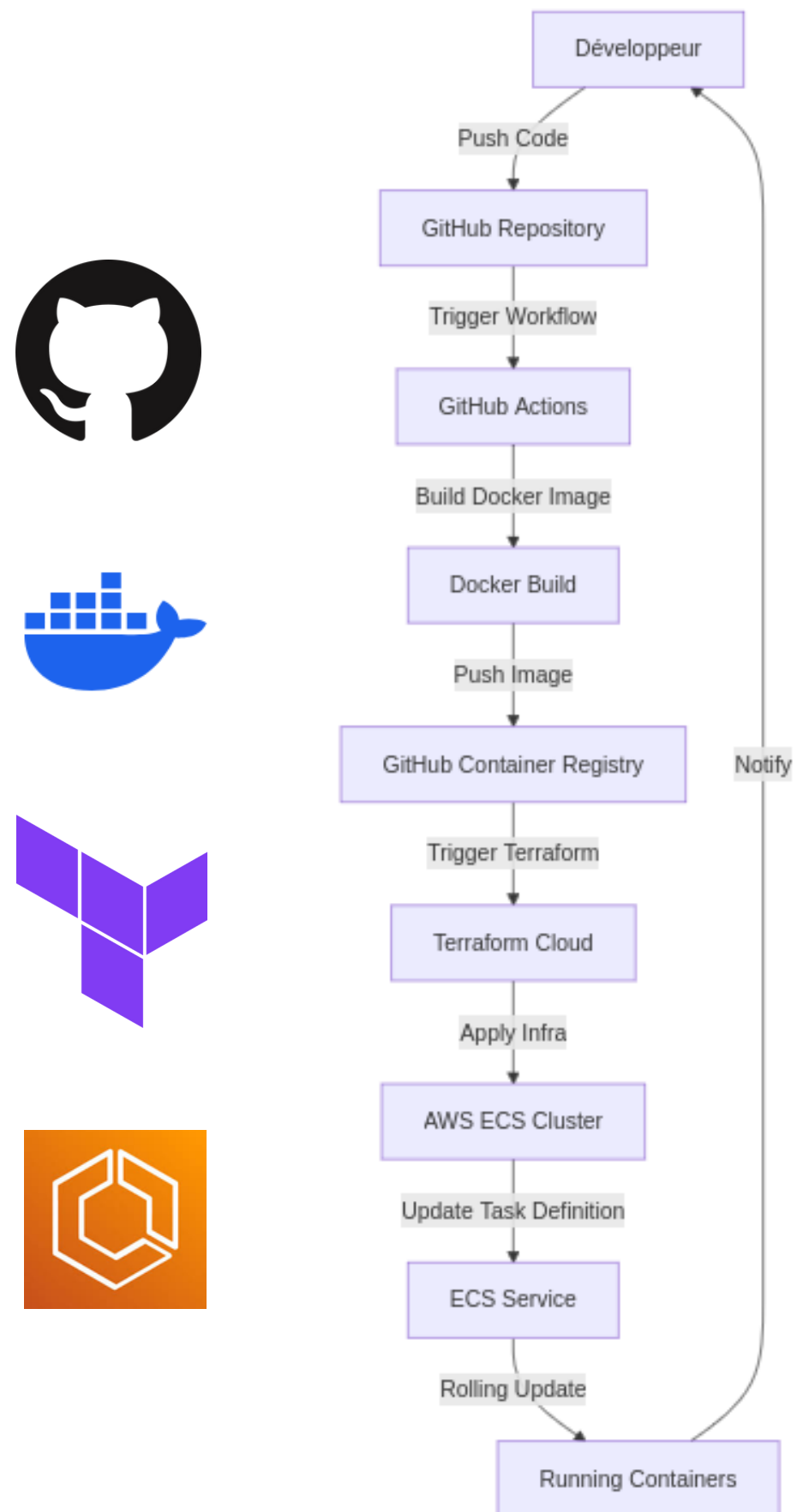
  deployment_minimum_healthy_percent = 50
  deployment_maximum_percent         = 200

  enable_execute_command = true
  launch_type             = "FARGATE"

  load_balancer {
    target_group_arn = var.target_group_arn
    container_name   = "std-ecs-chat"
    container_port    = 3000
  }

  network_configuration {
    subnets          = var.public_subnets
    security_groups    = [aws_security_group.chat.id]
    assign_public_ip   = true
  }
}
```

Workflow CI/CD



Demo

- Déploiement sur GitHub Actions
- Déploiement sur ECS

