ECS Service discovery workshop

In this workshop you're going to manually create and deploy 2 services into **ECS Fargate**, and configure **ECS Service Discovery** using Cloudmap so they can find and talk to each other.

The **backend** polls the service discovery API for information about the available namespaces, services and instances, and renders the result in an HTML page.

The **worker** application is a simple webserver, which responds to a *GET* '\' with a message.

Prerequisites

Clone the project git repository in your **Cloud9** environment.

```
git clone https://github.com/santatamas/ecs-workshop.git
cd ~/environment/ecs-workshop
```

Creating the ECR repositories

First, create 2 docker image repositories for our backend and worker images.

```
aws ecr create-repository --repository-name flask-backend
aws ecr create-repository --repository-name flask-worker
```

Setting up the required environment variables

You need to export a few environment variables for later use.

```
export BACKEND_REPO=$(aws ecr describe-repositories | \
jq -r .repositories[0].repositoryUri)
export WORKER_REPO=$(aws ecr describe-repositories | \
jq -r .repositories[1].repositoryUri)
```

Building the container images

Next, you'll build the docker images for the worker, and backend applications.

```
cd backend; docker build . -t flask-backend:latest; cd ..
cd workers; docker build . -t flask-worker:latest; cd ..
```

Tagging & pushing the images to ECR

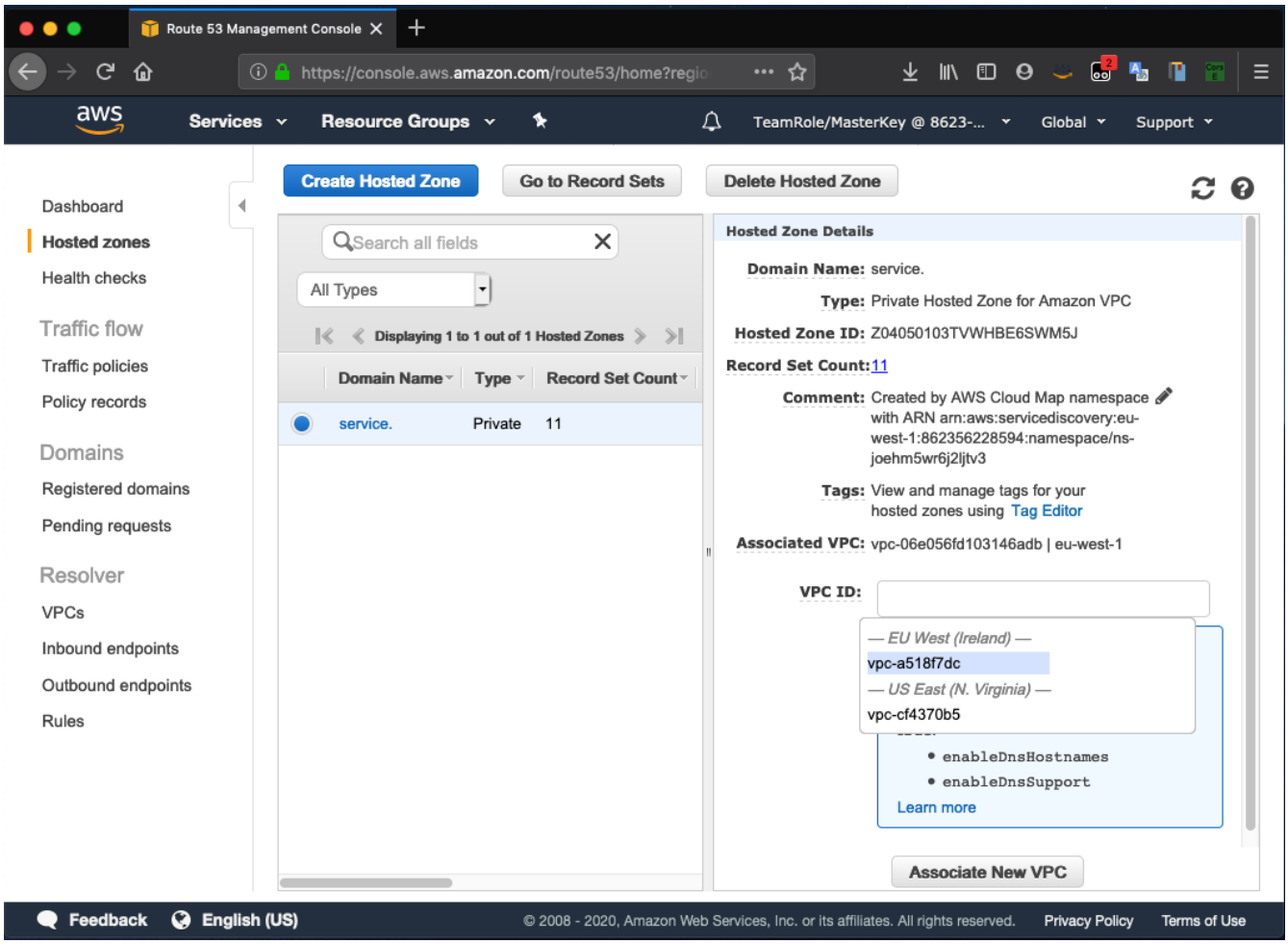
As a final step, let's tag the images, and push them to the ECR repositories you just created.

```
$(aws ecr get-login --no-include-email --region eu-west-1)
docker tag flask-backend:latest ${BACKEND_REPO}:latest
docker push ${BACKEND_REPO}:latest
docker tag flask-worker:latest ${WORKER_REPO}:latest
docker push ${WORKER_REPO}:latest
```

[OPTIONAL] Running the backend application from Cloud9

It's possible to run the **backend** application straight from your Cloud9 environment. By default, the hosted zone in Route53 is not associated with your Cloud9 VPC, hence the private DNS name resolution for your services will fail.

To fix this, open your AWS Management Console, and navigate to Route53. Select your hosted zone (should be called **service.**), and from the VPC ID dropdown on the right hand side, select the VPC of your Cloud9 environment.



Finally, click on *Associate New VPC*.

Export your AWS credentials to environment variables

```
export AWS_ACCESS_KEY_ID=$(aws configure get default.aws_access_key_id)
export AWS_SECRET_ACCESS_KEY=$(aws configure get default.aws_secret_access_key)
export AWS_SESSION_TOKEN=$(aws configure get default.aws_session_token)
```

Run the following command in your Cloud9 Terminal to run the **backend** docker image locally:

```
docker run -p 80:80 \
-e AWS_ACCESS_KEY_ID=${AWS_ACCESS_KEY_ID} \
-e AWS_SECRET_ACCESS_KEY=${AWS_SECRET_ACCESS_KEY} \
-e AWS_DEFAULT_REGION=eu-west-1 \
-e AWS_SESSION_TOKEN=${AWS_SESSION_TOKEN} \
-d flask-backend:latest
```

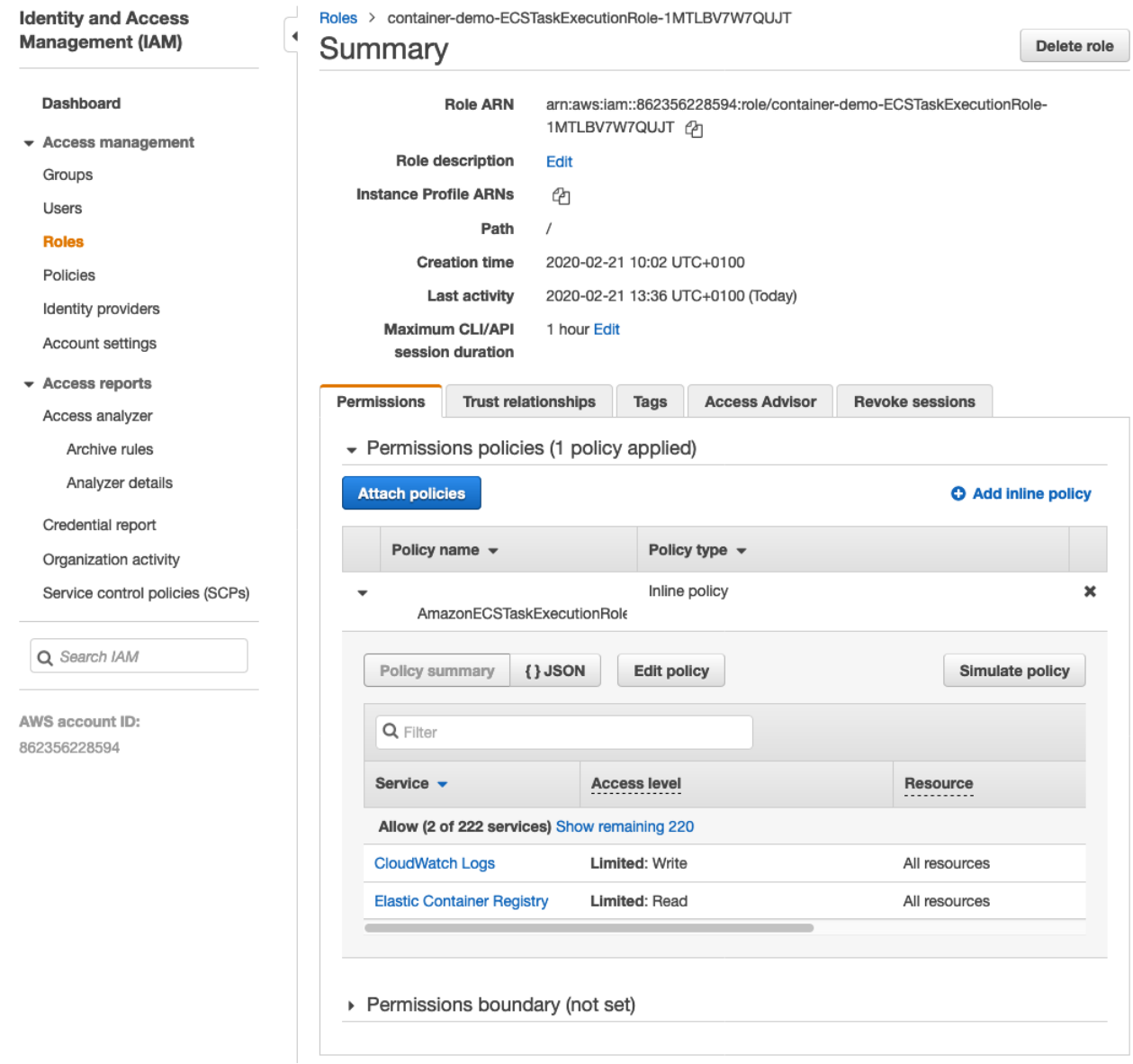
Now you should be able to access the website within your Cloud9 environment, without having to deploy it to ECS.

```
curl localhost
```

Allowing the ECS Task execution role to access the Service Discovery API

Our backend application will access the Service Discovery API to retrieve a list of namespaces, services and instances. The Task Execution role we created during the previous workshop does not provide access to this API, so we'll have to extend it.

- Open IAM, and select the Task Execution role starting with **container-demo-ECSTaskExecutionRole**



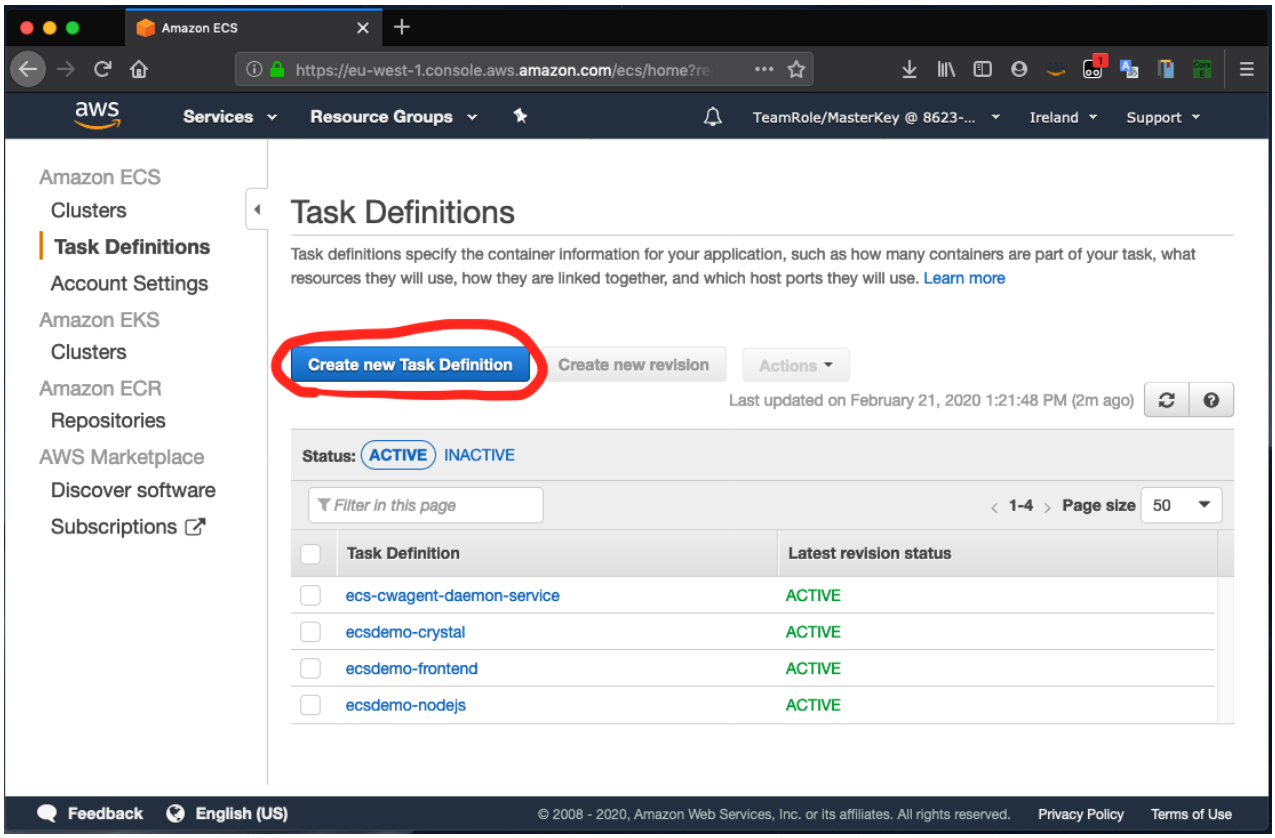
- Select the existing policy, and update it with the following JSON

```
{
```

```
  "Version": "2012-10-17",
  "Statement": [
      {
          "Sid": "VisualEditor0",
          "Effect": "Allow",
          "Action": [
              "servicediscovery:*",
              "logs:CreateLogStream",
              "ecr:GetDownloadUrlForLayer",
              "ecr:BatchGetImage",
              "ecr:GetAuthorizationToken",
              "logs:PutLogEvents",
              "ecr:BatchCheckLayerAvailability"
          ],
          "Resource": "*"
      }
  ]
}
```

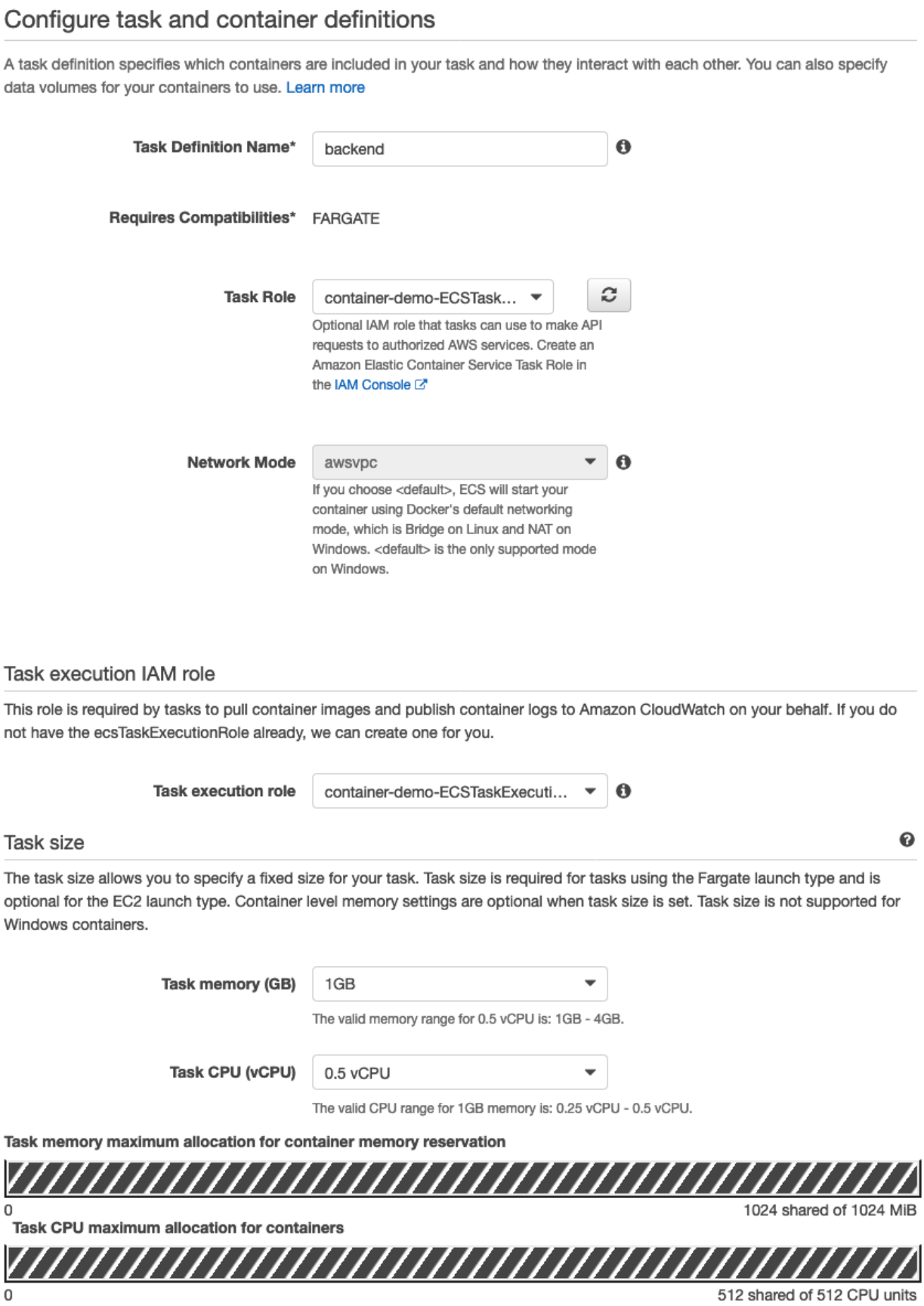
Creating a new ECS Task Definition for the backend

- Open your AWS Management Console, and navigate to Amazon ECS.
- Create a new Task Definition

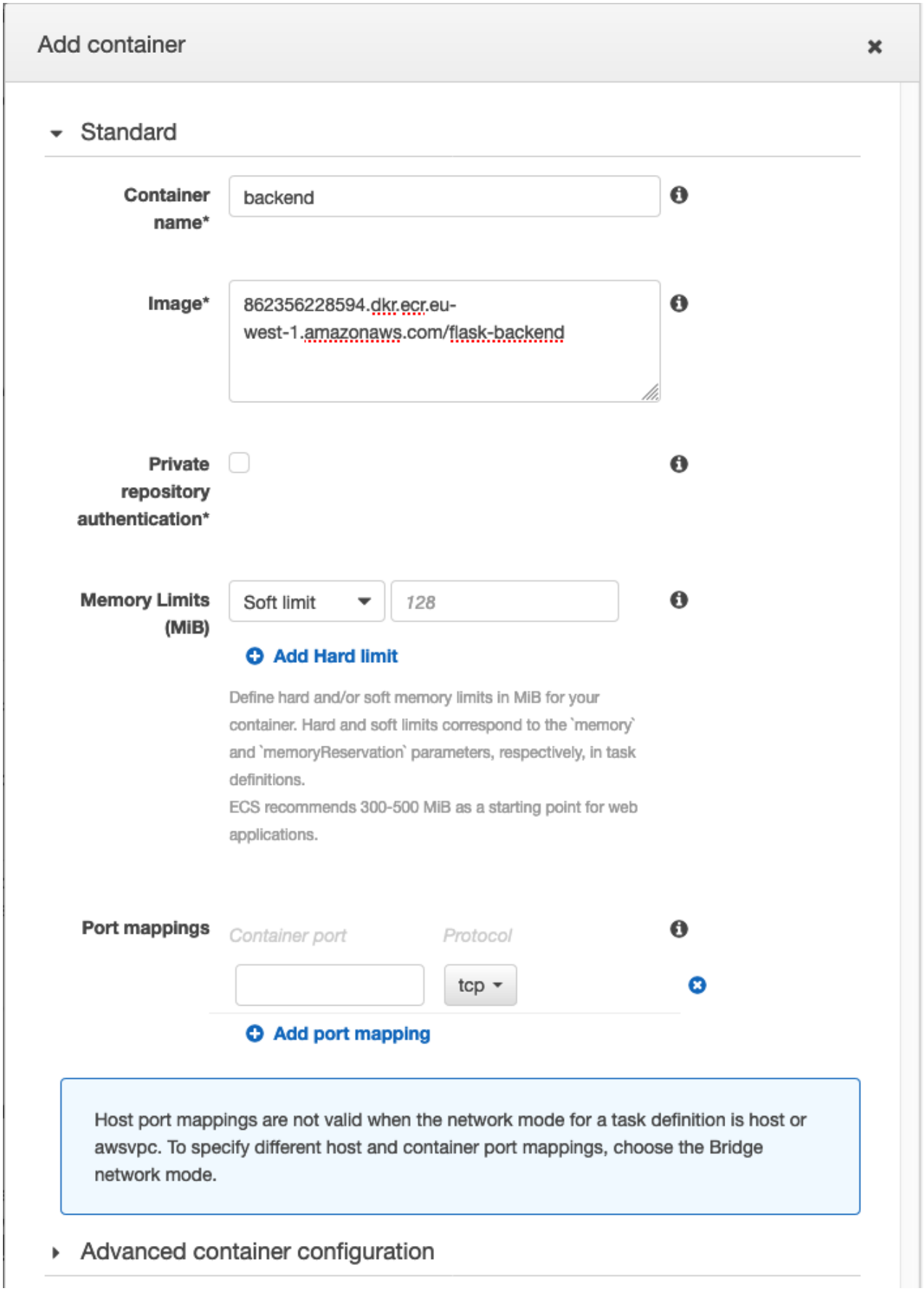


- Select **FARGATE**, and click *Next step*

- Next, name your definition *backend*, and re-use the task and execution roles from your previous workshop. For task memory and CPU, use 1GB and 0.5 vCPU.



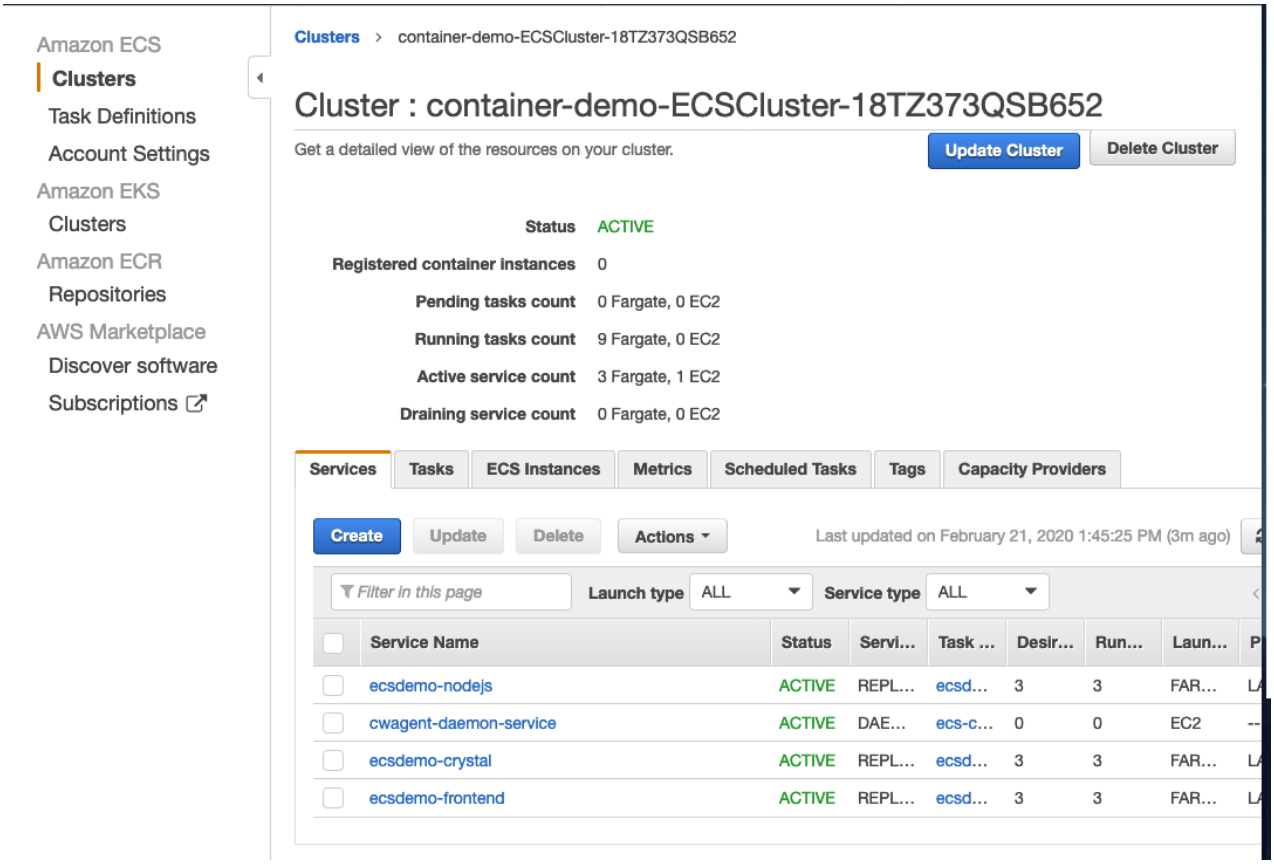
- Add a new container definition, and use the URI of the **backend** docker image from your ECR repository.



- Finally, click "Create" to create the Task definition.

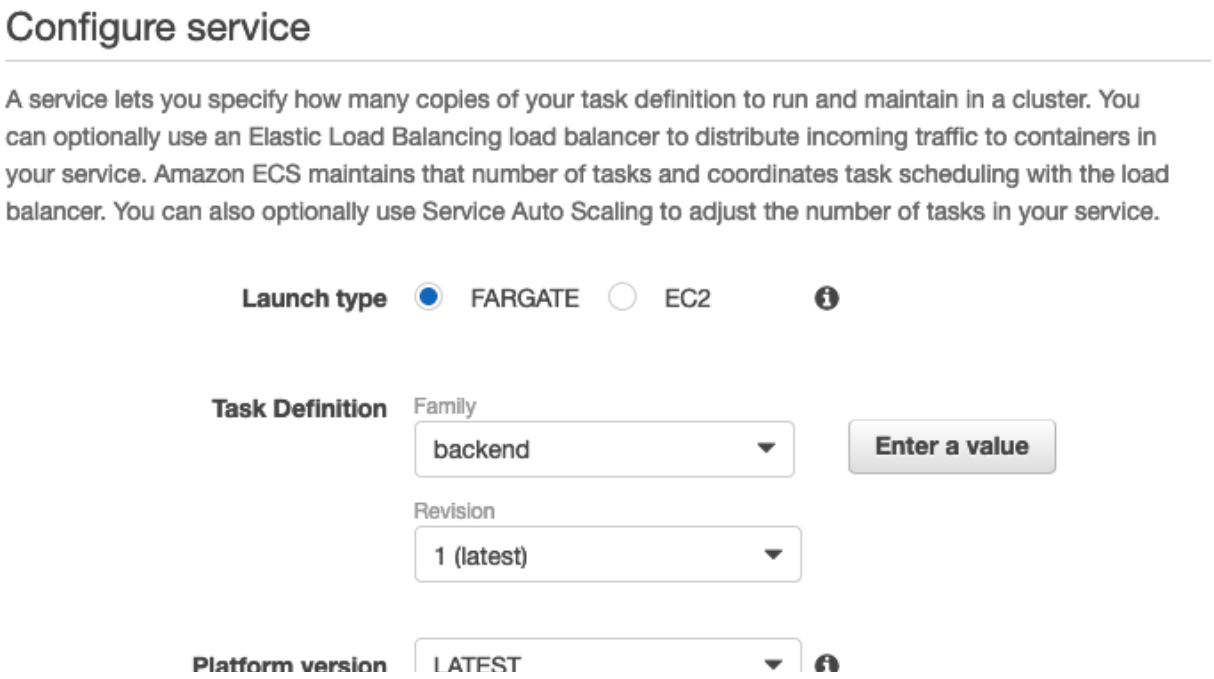
Creating a new ECS Service for the backend

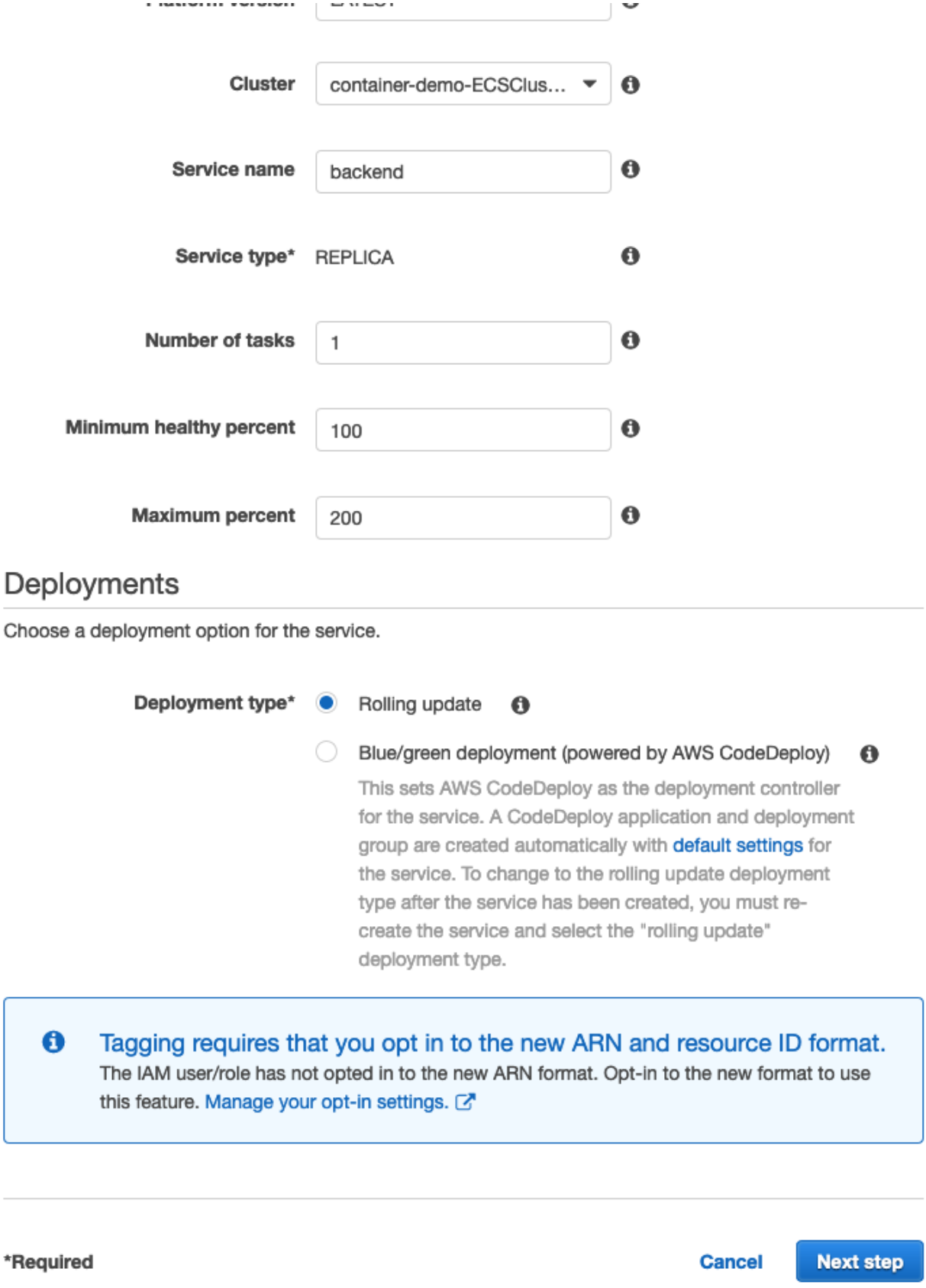
- Open Amazon ECS, and display the Clusters



- Click *Create* to create a new Service

- Use the values shown below, then click *Next step*





- Use the existing VPC and subnet for the service. Make sure to select a public subnet, you'll have to be able to access the container from your browser.

Configure network

VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC* vpc-06e056fd103146adb ...

Subnets* subnet-0e02a7d77c1ead58e (10.0.0.0/24) - eu-west-1a assign ipv6 on creation: Disabled

Security groups* backen-5404 **Edit**

Auto-assign public IP ENABLED

Health check grace period

If your service's tasks take a while to start and respond to ELB health checks, you can specify a health check grace period of up to 2,147,483,647 seconds during which the ECS service scheduler will ignore ELB health check status. This grace period can prevent the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up. This is only valid if your service is configured to use a load balancer.

Health check grace period requires a load balancer.

- Use the existing Service Discovery namespace (created during the previous workshop)

Service discovery (optional)

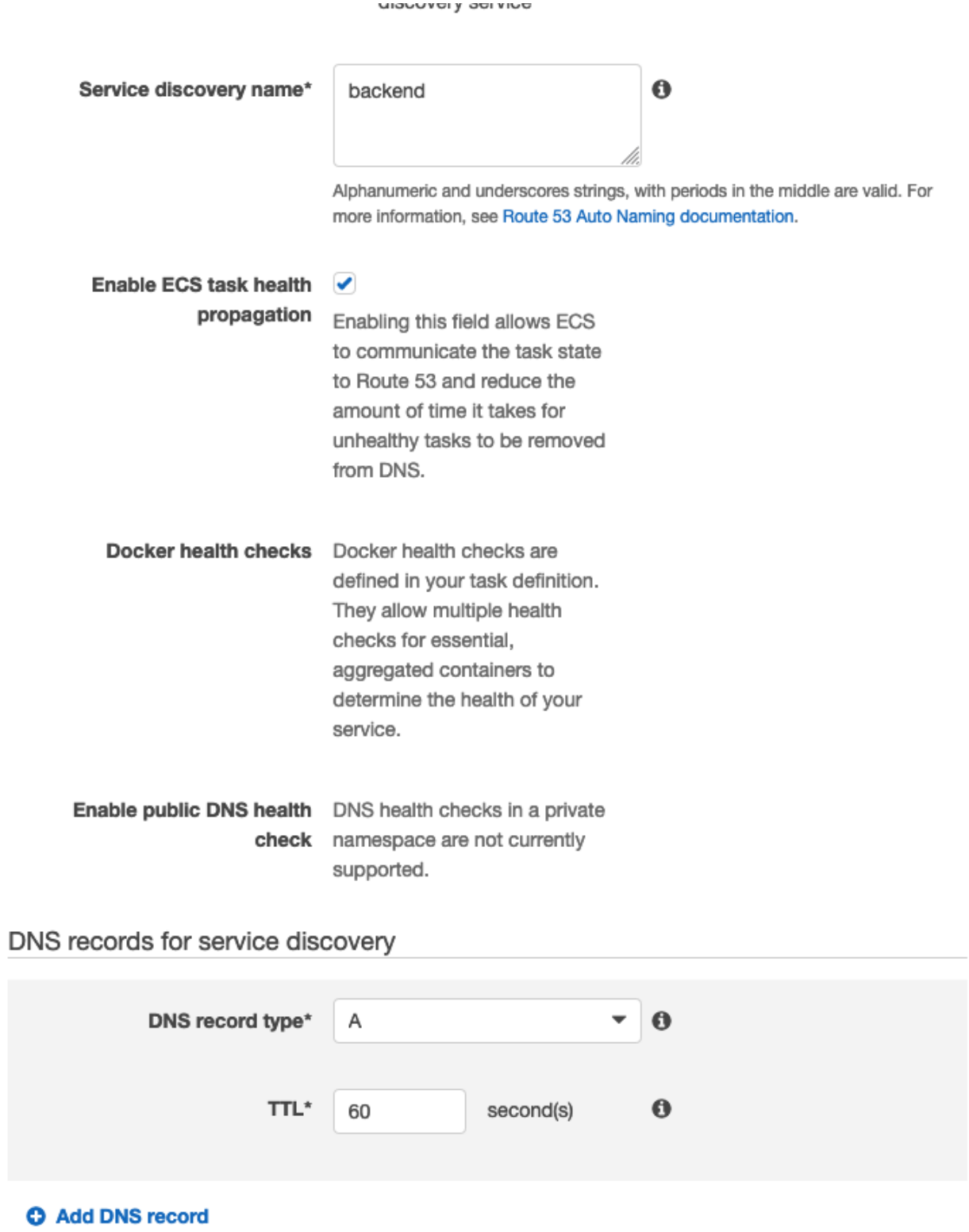
Service discovery uses Amazon Route 53 to create a namespace for your service, which allows it to be discoverable via DNS.

Enable service discovery integration ☑

Namespace* service | PRIVATE

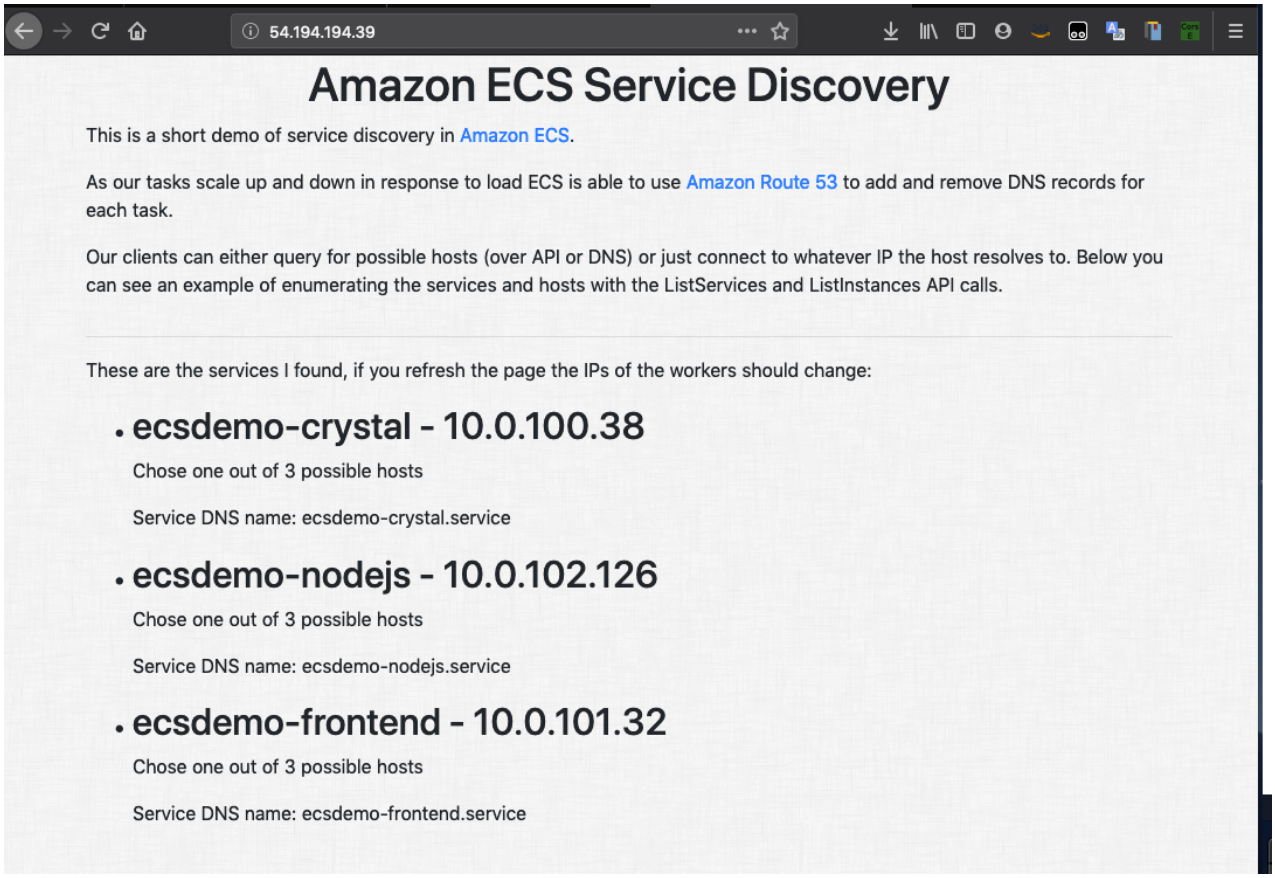
Configure service discovery service

- ◉ Create new service discovery service
- ○ Select existing service discovery service



- Finish the setup, and create the service (Next->Next->Finish)

- Open the public IP of the container, and you should see the running application

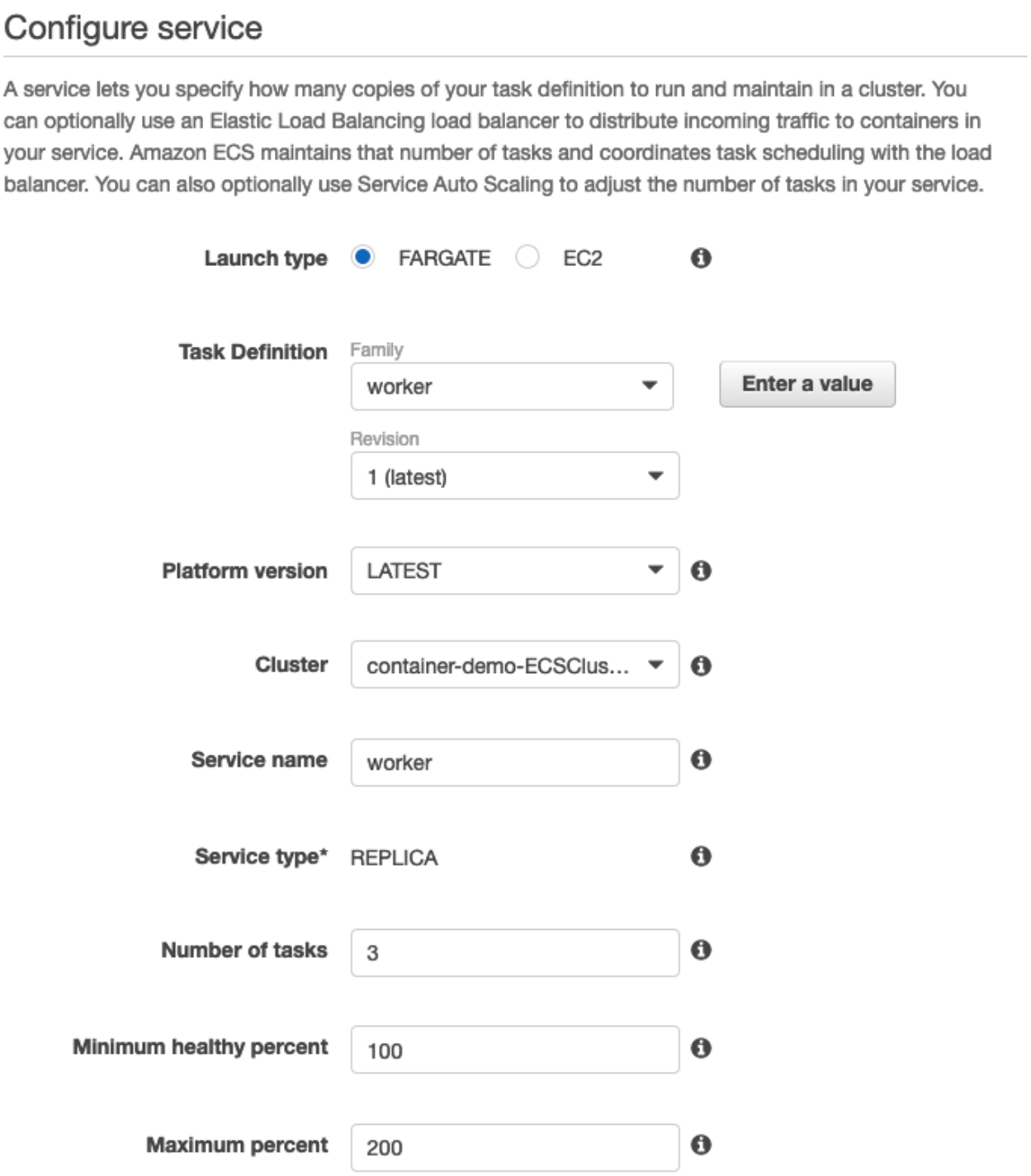


Creating a new ECS Task Definition for the worker nodes

- Create a new Task Definition using the previous steps. Use the name "worker", and use the URI of the **worker** docker image from your ECR repository.

Creating a new ECS Service for the workers

- Create a new service
- Use the settings shown below. Please note that we specified 3 running instances for this service, to test load balancing.



- Use the VPC settings shown below. This time, try to specify a different subnet for the worker instances.

Configure network

VPC and security groups

VPC and security groups are configurable when your task definition uses the awsvpc network mode.

Cluster VPC* vpc-06e056fd103146adb ...

Subnets* subnet-02811583cc53c7a5b (10.0.102.0/24) - eu-west-1c assign ipv6 on creation: Disabled

Security groups* worker-2269 **Edit**

Auto-assign public IP ENABLED

Health check grace period

If your service's tasks take a while to start and respond to ELB health checks, you can specify a health check grace period of up to 2,147,483,647 seconds during which the ECS service scheduler will ignore ELB health check status. This grace period can prevent the ECS service scheduler from marking tasks as unhealthy and stopping them before they have time to come up. This is only valid if your service is configured to use a load balancer.

Health check grace period requires a load balancer.

- Use the existing Service Discovery Namespace (created during the previous workshop)

Service discovery (optional)

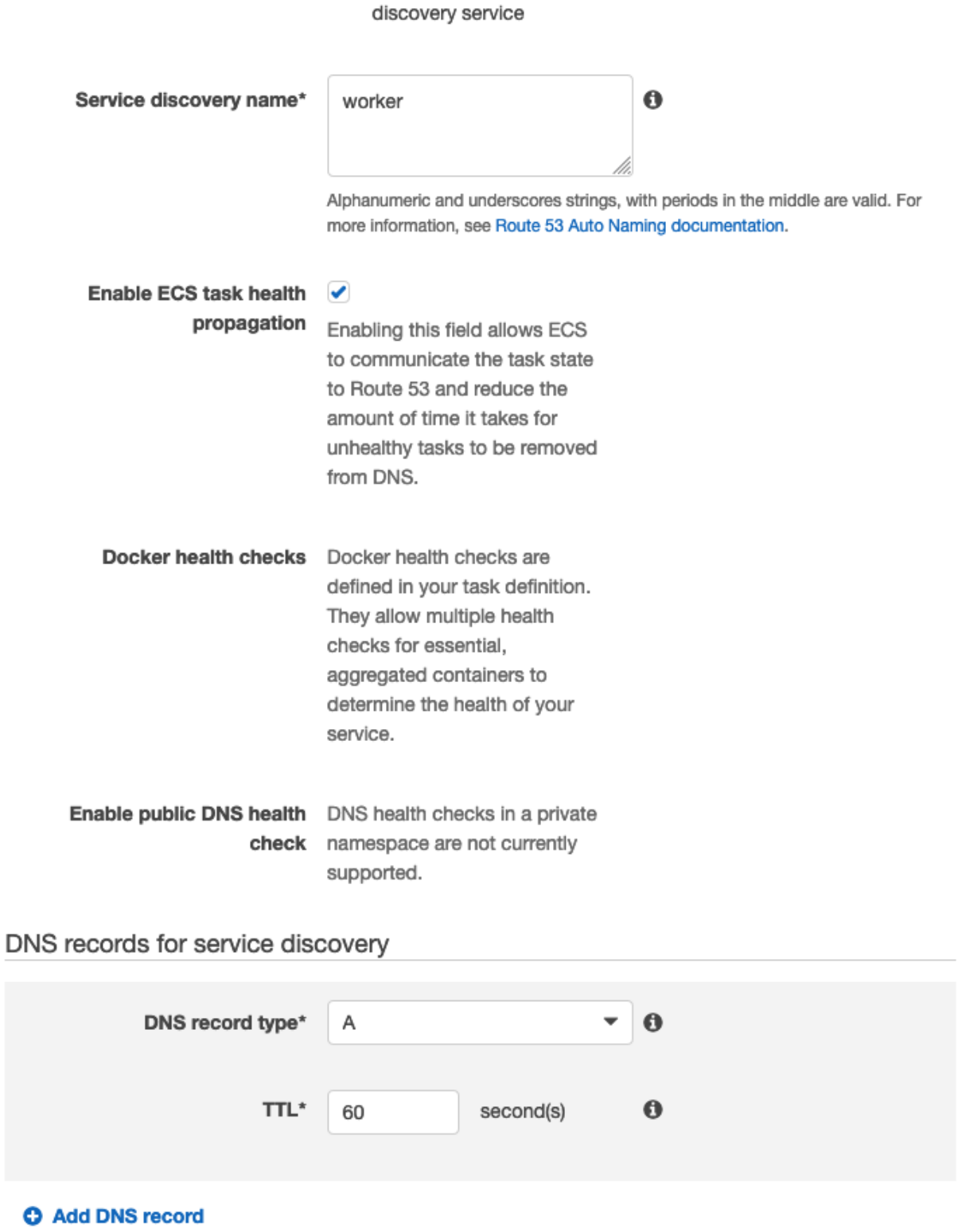
Service discovery uses Amazon Route 53 to create a namespace for your service, which allows it to be discoverable via DNS.

Enable service discovery integration ☑

Namespace* service | PRIVATE

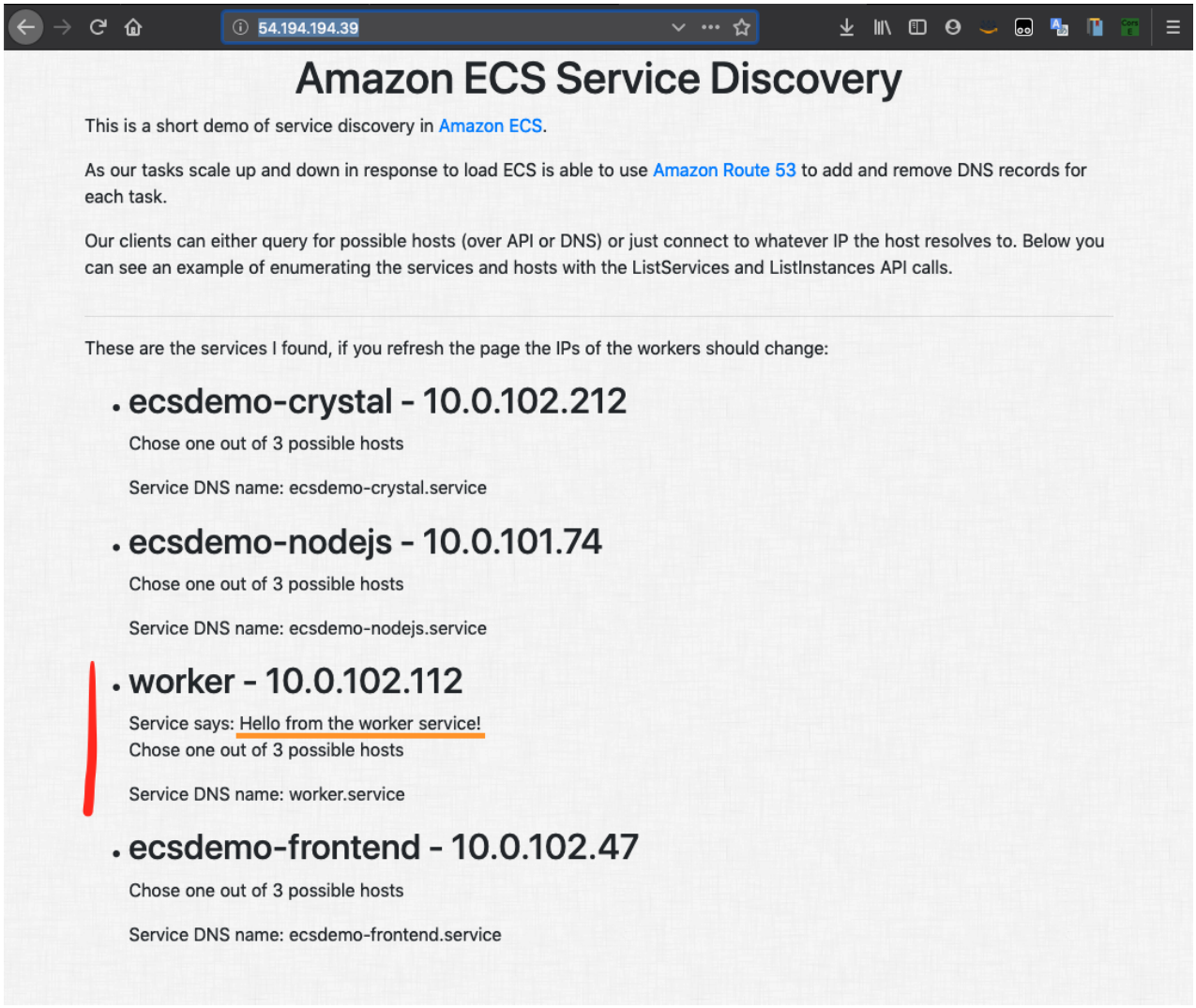
Configure service discovery service

- (•) Create new service discovery service
- ( ) Select existing service



- Refresh the **backend** application in your browser to see the newly discovered worker service. The application also sends an HTTP request to the service, and displays the

response.



Summary

In this exercise you:

- Set up ECR repositories, built and pushed docker images
- Created ECS Fargate Task Definitions for the **backend** and **worker** applications
- Deployed the Task Definitions to services
- Used service discovery to communicate with the deployed services