😊 AMAZON ECR (ELASTIC CONTAINER REGISTRY)

Amazon ECR (Amazon Elastic Container Registry) is a fully managed container registry service provided by Amazon Web Services (AWS). It is designed to store, manage, and deploy Docker container images. Docker containers are a lightweight and portable way to package and run applications along with their dependencies.

Here are some key features and functionalities of Amazon ECR:

1. **Secure and Private Repositories:** ECR allows you to create private Docker repositories to securely store and manage your container images. Access to these repositories can be controlled using AWS Identity and Access Management (IAM) policies.
2. **Integration with AWS Services:** ECR seamlessly integrates with other AWS services, such as Amazon ECS (Elastic Container Service) and Kubernetes on AWS (via Amazon EKS - Elastic Kubernetes Service), making it easier to deploy and manage containerized applications.
3. **Scalability:** ECR is designed to scale with your container image storage needs. It provides automatic scaling to handle increased demand and ensures high availability and durability of your container images.
4. **Lifecycle Policies:** You can define lifecycle policies to automatically clean up unused or expired images. This helps in managing storage costs and ensures that only the necessary images are retained.
5. **Container Image Encryption:** ECR supports encryption of container images at rest to enhance security. Images are encrypted using AWS Key Management Service (KMS) keys.
6. **Docker CLI Integration:** ECR can be easily accessed and managed using the Docker CLI, making it familiar for developers already accustomed to Docker.

😄 USE CASE OF AMAZON ECR:

Amazon ECR (Elastic Container Registry) is commonly used in various scenarios where organizations leverage containerization technologies, such as Docker, and need a secure and scalable solution for storing, managing, and deploying container images. Here are some common use cases for Amazon ECR:

1. **Containerized Application Deployment:**

- Organizations that have adopted a microservices architecture or containerized applications can use ECR to store and manage their Docker container images.
- ECR integrates seamlessly with Amazon ECS (Elastic Container Service) and Amazon EKS (Elastic Kubernetes Service), making it easier to deploy and manage containerized applications in a scalable and efficient manner.

2. **Continuous Integration and Continuous Deployment (CI/CD):**

- ECR is often integrated into CI/CD pipelines to store and version Docker images produced during the build process.

- CI/CD tools, such as AWS Code Pipeline, Jenkins, or GitLab CI, can push newly built container images to ECR, and then trigger the deployment of updated containers in the target environment.

3. **Private Image Repositories:**

- ECR provides private Docker image repositories, allowing organizations to securely store and manage their container images without exposing them to the public.
- This is particularly important for sensitive or proprietary applications, where access control and security are paramount.

4. **Multi-Region Deployment:**

- Organizations with a global presence or those looking for disaster recovery options can use ECR to replicate container images across multiple AWS regions.
- This ensures that containerized applications can be deployed in different regions, providing low-latency access and high availability.

5. **Integration with AWS Services:**

- ECR integrates seamlessly with other AWS services, making it easy to incorporate containerized applications into a broader AWS ecosystem.
- For example, ECR can be used in conjunction with AWS Fargate for serverless container orchestration or with AWS Lambda for running containerized workloads as part of a serverless architecture.

6. **Security and Compliance:**

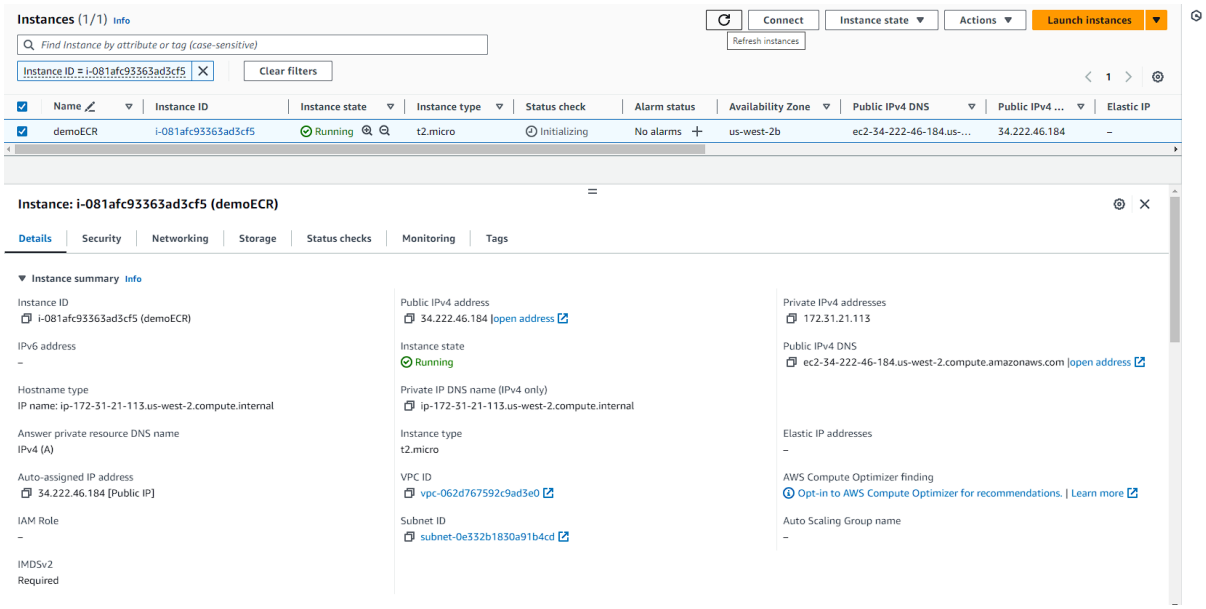
- ECR provides features such as image encryption and IAM-based access controls, which are crucial for meeting security and compliance requirements.
- The ability to manage access to container images using IAM roles allows organizations to enforce fine-grained access control over who can pull or push images.

7. **Scalable and Reliable Image Storage:**

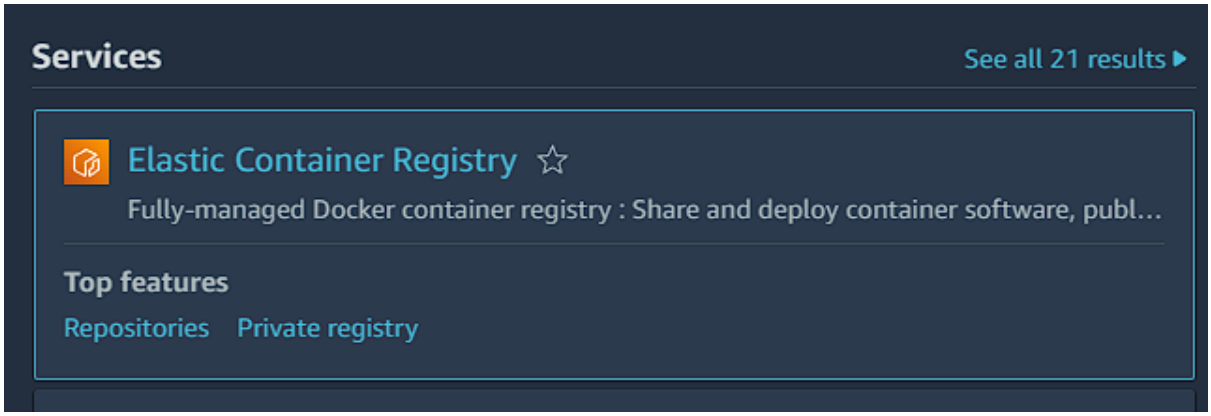
- ECR automatically scales to accommodate increasing storage needs for container images.
- With high availability and durability, ECR ensures that container images are reliably stored and can be quickly retrieved when needed for deployment.

🤓 TO BEGIN WITH THE LAB:

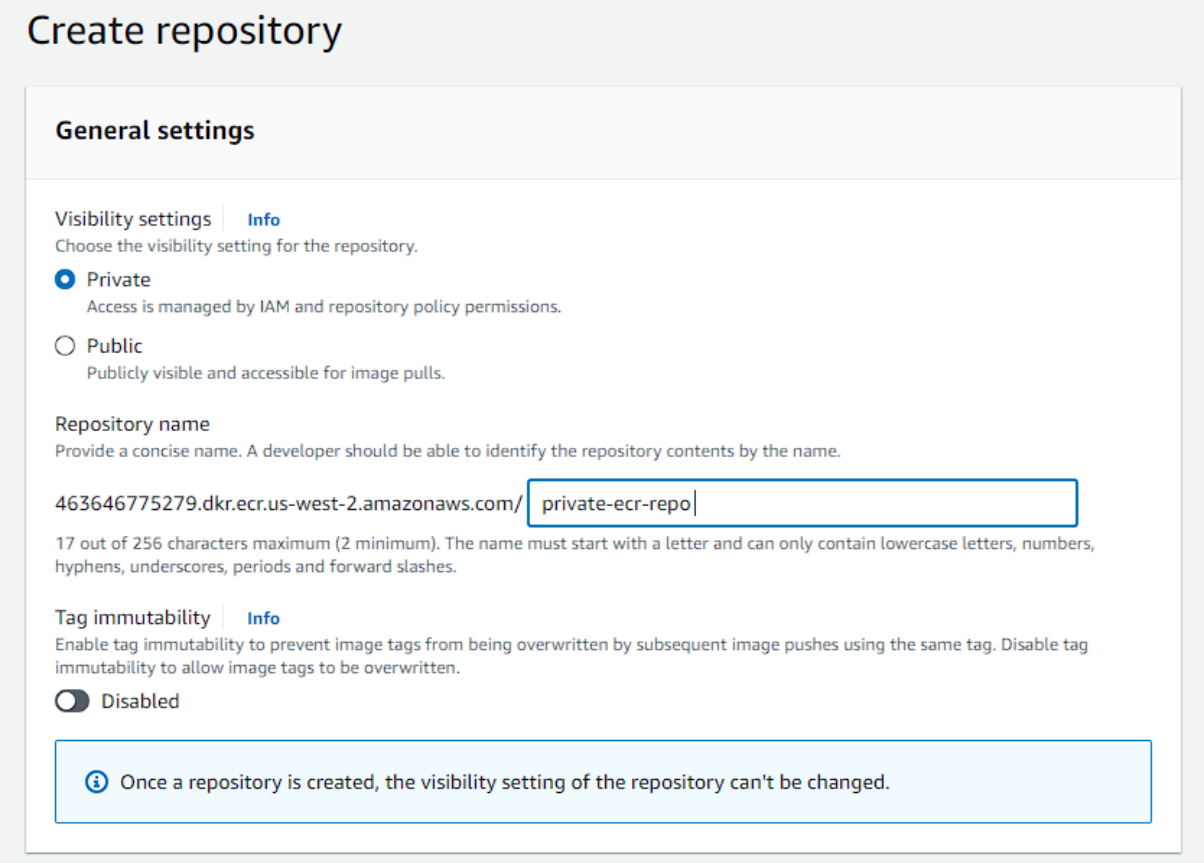
1. Log in to Amazon console. Then search EC2.
2. After that you are needed to create an EC2 instance with Linux 2 AMI.
3. So, this EC2 instance will be used to set up the docker and the image will be pushed onto it.

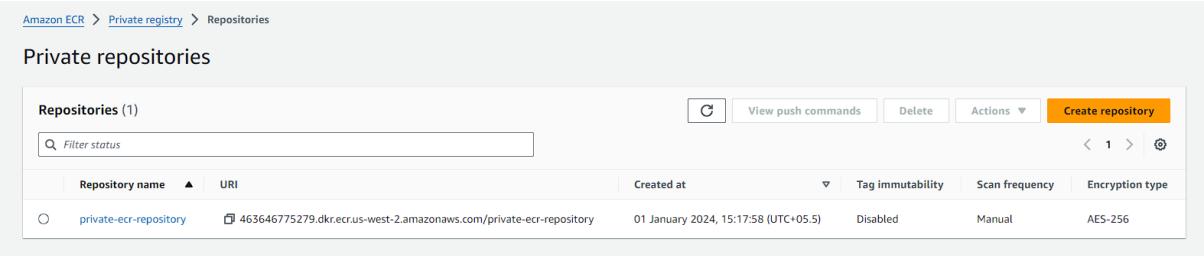


4. Now you need to search ECR service. Open it.
5. Here you need to create a repository.

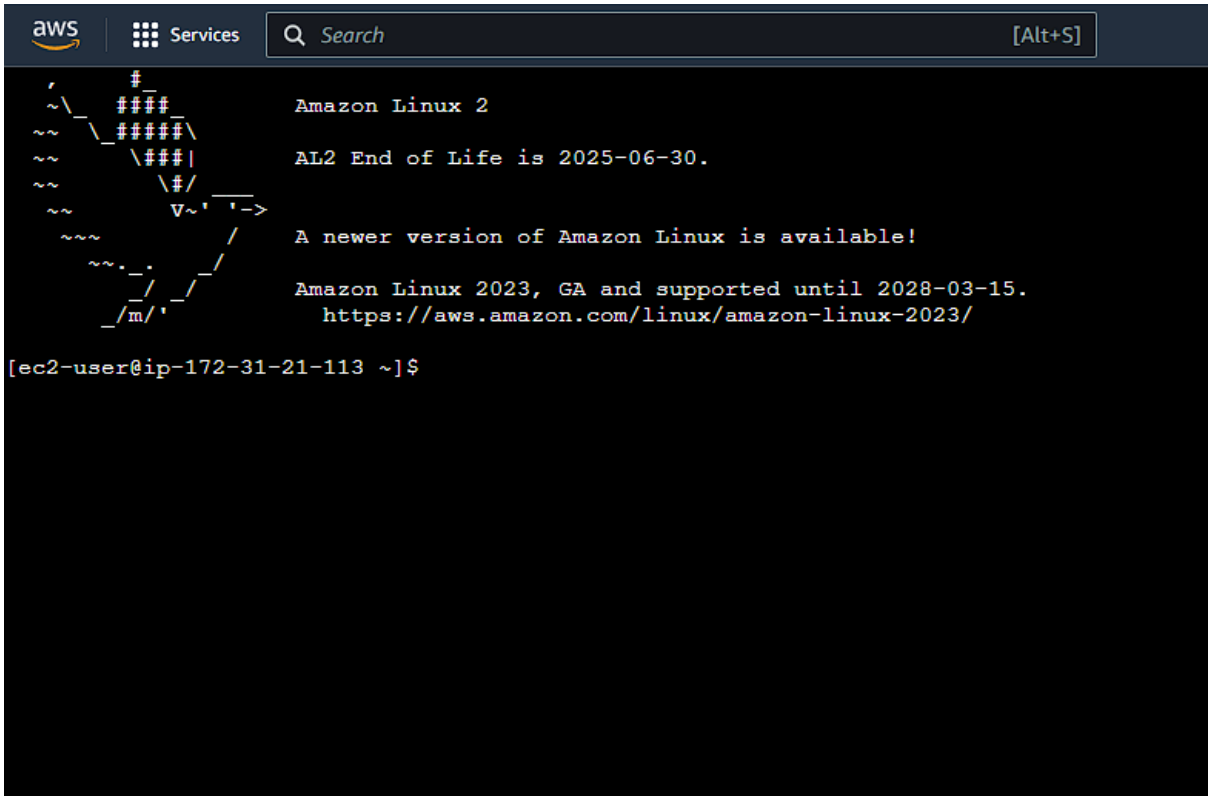


6. Once you have clicked on Get Started, you'll be on a new page to create your private repository.
7. Now keep it private and give it a name.
8. Then scroll down keep everything to private, and create your repository.

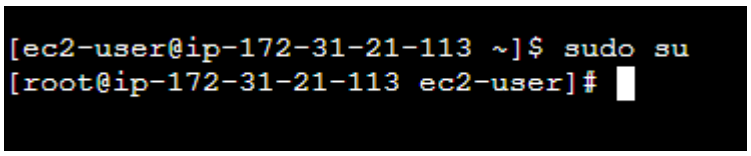




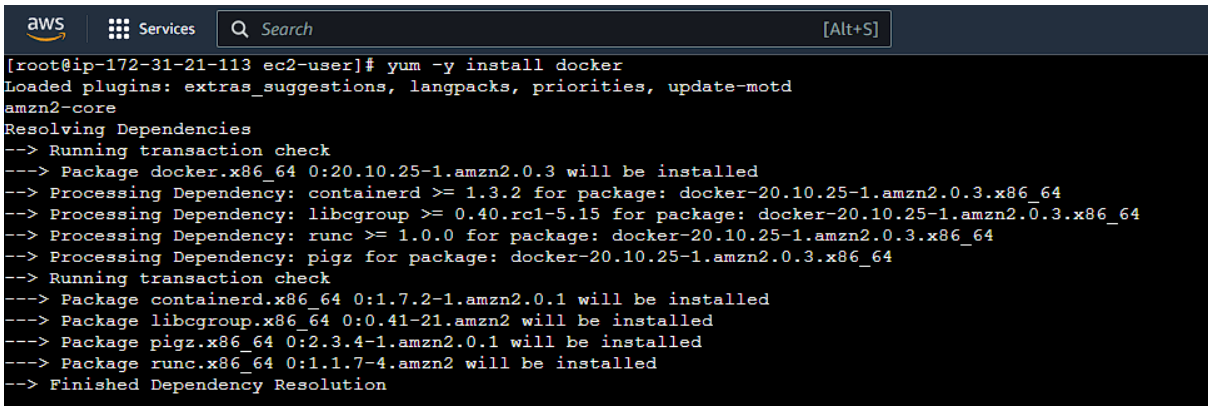
9. Now the repository is created all you need to do is to push the container image to this repo.
10. For that quickly go back to your instance and connect to it.



11. Now run a command **sudo su,** so that you can access the root user.



12. Now you need to install docker. In Linux 2 AMI the installation of docker is quite simple you just need to run this command.
    Yum -y install docker



13. Once the docker is installed, and you want to check that whether it is running or not. Then you can run this command.
    service docker status
14. So, as you can see currently you docker is inactive/dead.

```
[root@ip-172-31-21-113 ec2-user]# service docker status
Redirecting to /bin/systemctl status docker.service
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: inactive (dead)
     Docs: https://docs.docker.com
[root@ip-172-31-21-113 ec2-user]#
```

15. There are some commands through which you can start the docker.
16. The first command will start docker, and the second command will show you the status of the docker.

    Systemctl start docker

    Systemctl status docker

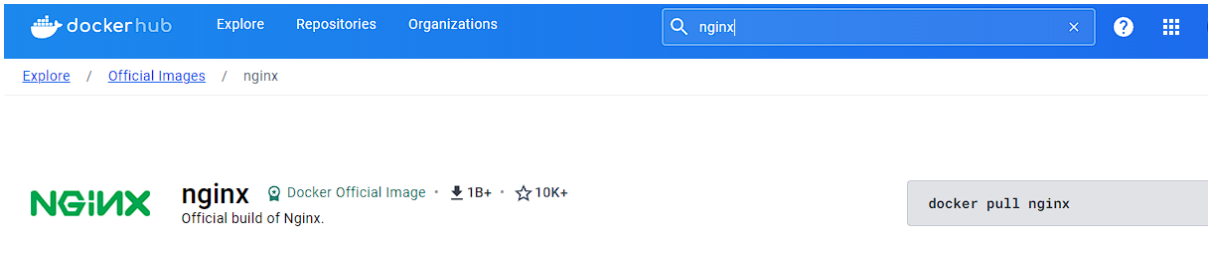
```
[root@ip-172-31-21-113 ec2-user]# systemctl start docker
[root@ip-172-31-21-113 ec2-user]# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; disabled; vendor preset: disabled)
   Active: active (running) since Mon 2024-01-01 10:01:09 UTC; 34s ago
     Docs: https://docs.docker.com
  Process: 3894 ExecStartPre=/usr/libexec/docker/docker-setup-runtimes.sh (code=exited, status=0/SUCCESS)
  Process: 3893 ExecStartPre=/bin/mkdir -p /run/docker (code=exited, status=0/SUCCESS)
 Main PID: 3897 (dockerd)
    Tasks: 7
   Memory: 22.9M
   CGroup: /system.slice/docker.service
           └─3897 /usr/bin/dockerd -H fd:// --containerd=/run/containerd/containerd.sock --default-ulimit nofile=32768:65536

Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.3692355502" level=info msg="ClientConn switching balancer to \"pick_first\"" module="grpc
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.4029690562" level=warning msg="Your kernel does not support cgroup blkio weight"
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.4034911342" level=warning msg="Your kernel does not support cgroup blkio weight_device"
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.4040006742" level=info msg="Loading containers: start."
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.5923954062" level=info msg="Default bridge (docker0) is assigned with an IP address 172.17.0...IP address"
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.6434879062" level=info msg="Loading containers: done."
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.6590555022" level=info msg="Docker daemon" commit=5df983c graphdriver(s)=overlay2 version=20.10.25
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.6596550842" level=info msg="Daemon has completed initialization"
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal systemd[1]: Started Docker Application Container Engine.
Jan 01 10:01:09 ip-172-31-21-113.us-west-2.compute.internal dockerd[3897]: time="2024-01-01T10:01:09.6983492672" level=info msg="API listen on /run/docker.sock"
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-172-31-21-113 ec2-user]#
```

17. So, if you just run docker then it will show you the basic health command.

```
[root@ip-172-31-21-113 ec2-user]# docker

Usage:  docker [OPTIONS] COMMAND

A self-sufficient runtime for containers

Options:
      --config string      Location of client config files (default "/root/.docker")
  -c, --context string     Name of the context to use to connect to the daemon (overrides DOCKER_HOST env var and default context set with "docker context use")
  -D, --debug              Enable debug mode
  -H, --host list          Daemon socket(s) to connect to
  -l, --log-level string   Set the logging level ("debug"|"info"|"warn"|"error"|"fatal") (default "info")
      --tls                Use TLS; implied by --tlsverify
      --tlscacert string   Trust certs signed only by this CA (default "/root/.docker/ca.pem")
      --tlscert string     Path to TLS certificate file (default "/root/.docker/cert.pem")
      --tlskey string      Path to TLS key file (default "/root/.docker/key.pem")
      --tlsverify          Use TLS and verify the remote
  -v, --version            Print version information and quit

Management Commands:
  builder     Manage builds
  buildx*     Docker Buildx (Docker Inc., v0.0.0+unknown)
  config      Manage Docker configs
  container   Manage containers
  context     Manage contexts
  image       Manage images
  manifest    Manage Docker image manifests and manifest lists
  network     Manage networks
  node        Manage Swarm nodes
  plugin      Manage plugins
  secret      Manage Docker secrets
  service     Manage services
  stack       Manage Docker stacks
  swarm       Manage Swarm
  system      Manage Docker
  trust       Manage trust on Docker images
  volume      Manage volumes
```

18. Now what you are going to do is you want to push a Docker container image into this repository. Now before you do that, you need to first have the image. So, you'll be pulling an image from the Docker Hub repository and then you'll be pushing it to ACR. So, the image that you'll be using for testing is nginX and the command to pull the image is.

    docker pull nginx

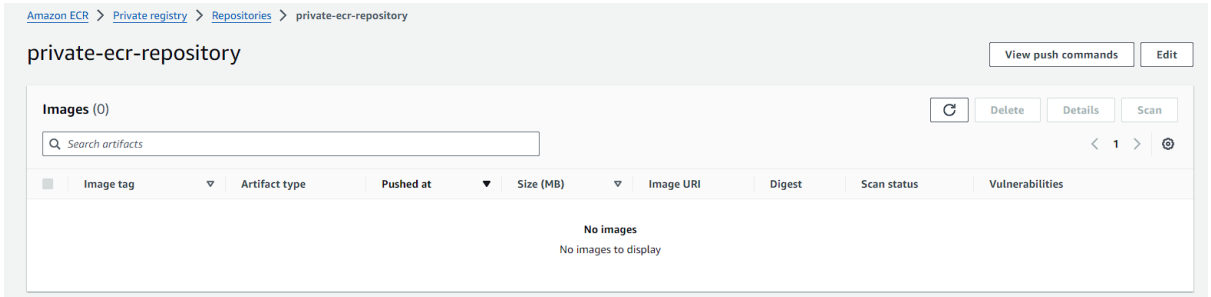



```
[root@ip-172-31-21-113 ec2-user]# docker pull nginx
Using default tag: latest
latest: Pulling from library/nginx
af107e978371: Pull complete
336ba1f05c3e: Pull complete
8c37d2ff6efa: Pull complete
51d6357098de: Pull complete
782f1ecce57d: Pull complete
5e99d351b073: Pull complete
7b73345df136: Pull complete
Digest: sha256:2bdc49f2f8ae8d8dc50ed00f2ee56d00385c6f8bc8a8b320d0a294d9e3b49026
Status: Downloaded newer image for nginx:latest
docker.io/library/nginx:latest
[root@ip-172-31-21-113 ec2-user]#
```

19. Now, if you do a **docker images** over here you will see that we have the engine X image that is currently downloaded in this specific EC2 instance.

```
[root@ip-172-31-21-113 ec2-user]# docker images
REPOSITORY   TAG       IMAGE ID       CREATED        SIZE
nginx        latest    d453dd892d93   2 months ago   187MB
[root@ip-172-31-21-113 ec2-user]#
```

20. So, now go back to your repository and open it. Basically, it will show you the push the commands. You just need to click on view push commands.



Push commands for private-ecr-repository

macOS / Linux | Windows

Make sure that you have the latest version of the AWS CLI and Docker installed. For more information, see Getting Started with Amazon ECR.

Use the following steps to authenticate and push an image to your repository. For additional registry authentication methods, including the Amazon ECR credential helper, see Registry Authentication.

1. Retrieve an authentication token and authenticate your Docker client to your registry.
   Use the AWS CLI:

   ```
   aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 463646775279.dkr.ecr.us-west-2.amazonaws.com
   ```

   Note: If you receive an error using the AWS CLI, make sure that you have the latest version of the AWS CLI and Docker installed.
2. Build your Docker image using the following command. For information on building a Docker file from scratch see the instructions here. You can skip this step if your image is already built:

   ```
   docker build -t private-ecr-repository .
   ```
3. After the build completes, tag your image so you can push the image to this repository:

   ```
   docker tag private-ecr-repository:latest 463646775279.dkr.ecr.us-west-2.amazonaws.com/private-ecr-repository:latest
   ```
4. Run the following command to push this image to your newly created AWS repository:

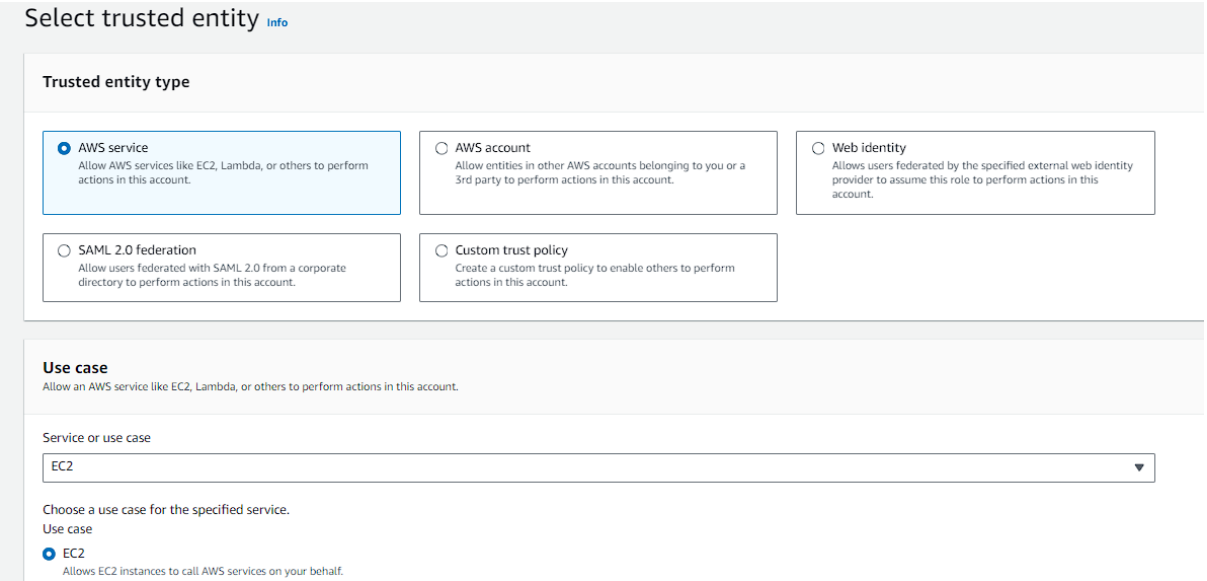
   ```
   docker push 463646775279.dkr.ecr.us-west-2.amazonaws.com/private-ecr-repository:latest
   ```

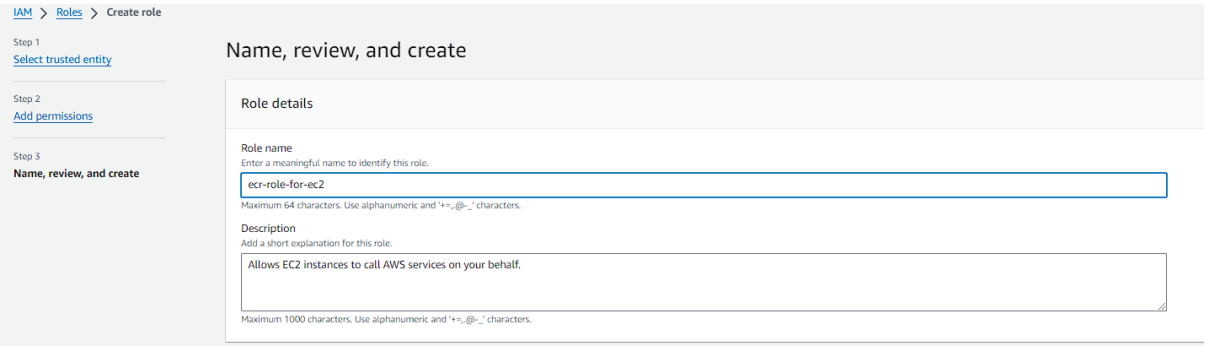
Close

21. So, when you will run the first the command it will say that unable to locate credentials.
22. So, what you need to do is either you can add the access and secret keys to this specific ec2 instance, or you can add an appropriate IAM rule. Let us go ahead and add an appropriate IAM rule.

```
[root@ip-172-31-21-113 ec2-user]# aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 463646775279.dkr.ecr.us-west-2.amazonaws.com
Unable to locate credentials. You can configure credentials by running "aws configure".
Error: Cannot perform an interactive login from a non TTY device
[root@ip-172-31-21-113 ec2-user]#
```

23. Go to IAM create a role without attaching any of the policies to it.

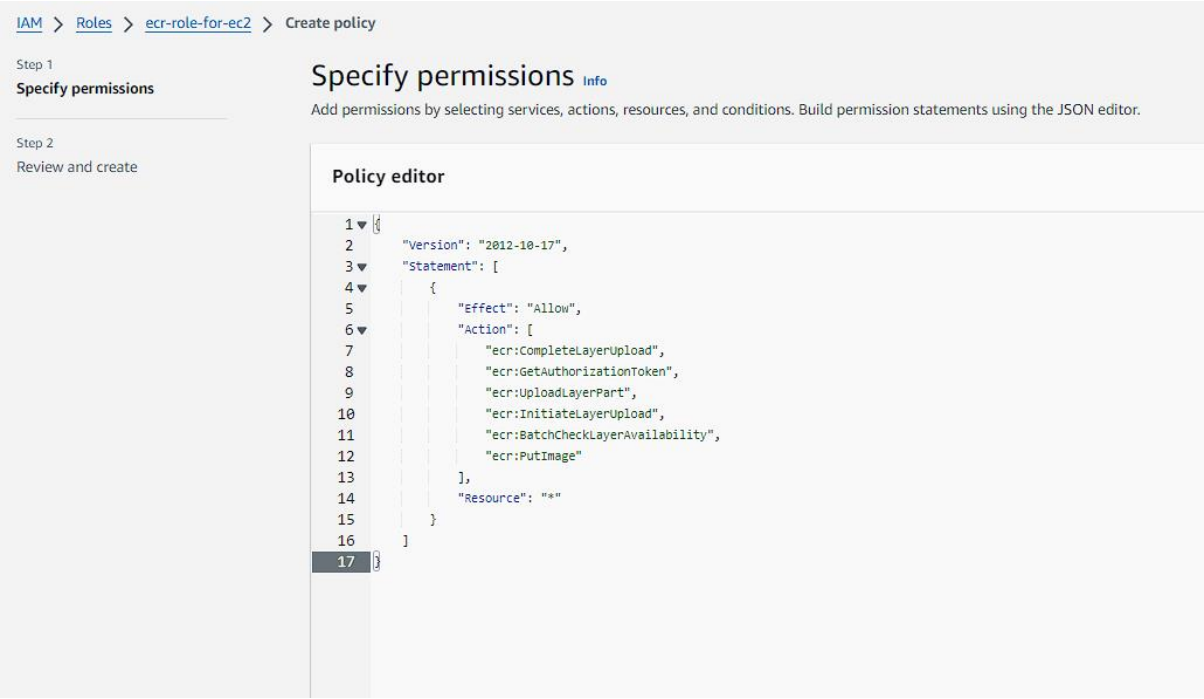


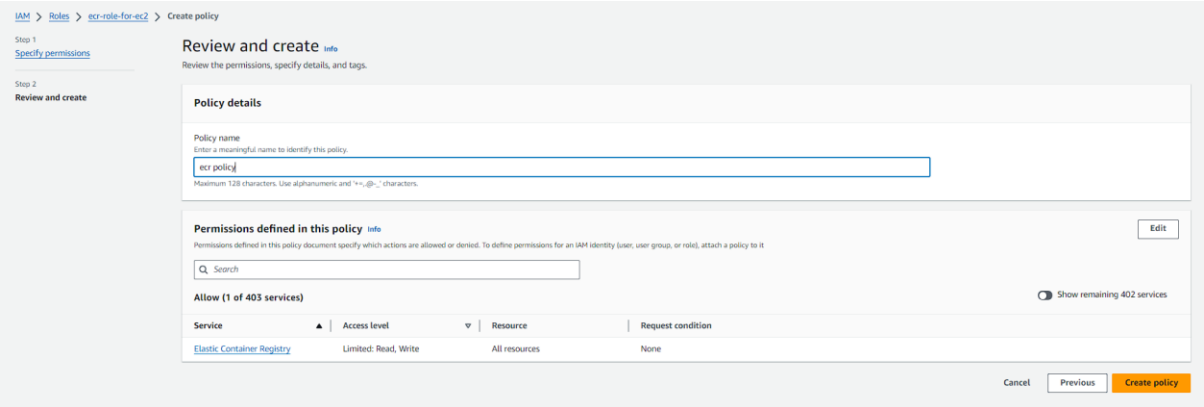


24. Once the role is created, then quickly open it up.
25. There you need to create an inline policy and attach it to the role.
26. You can the policy from amazon documentation, here is the link to the documentation. https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-push.html
27. Below is the policy that you are going to use. You can either use the link to get to the policy or you can use the policy mentioned below.

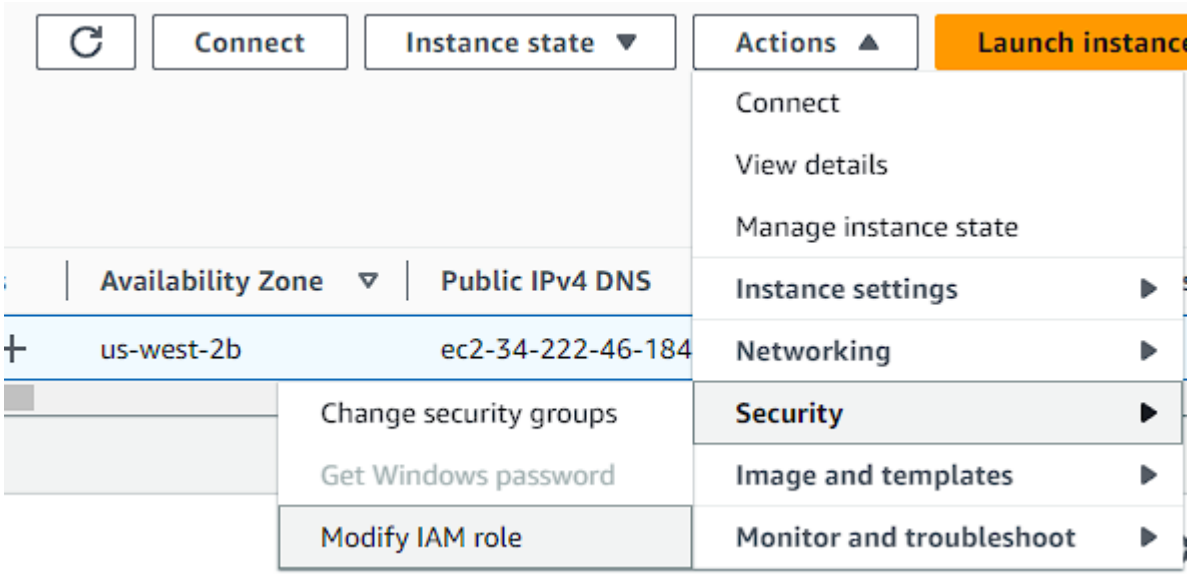
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ecr:CompleteLayerUpload",
        "ecr:GetAuthorizationToken",
        "ecr:UploadLayerPart",
        "ecr:InitiateLayerUpload",
```

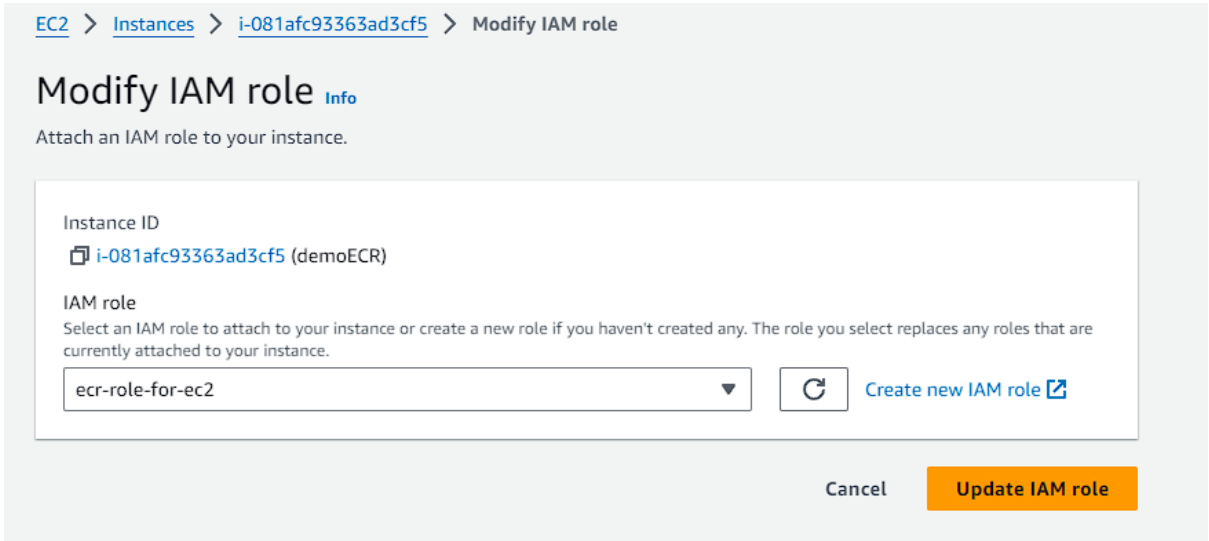
```
            "ecr:BatchCheckLayerAvailability",
            "ecr:PutImage"
        ],
        "Resource": "*"
    }
  ]
}
```



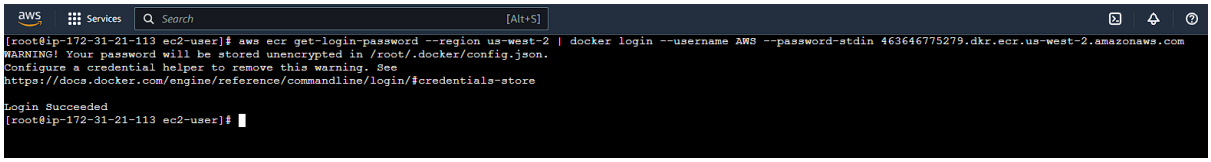


28. Once your role and policy are ready you need to attach this role and policy to ec2 instance.
29. For that go back to ec2 instance.
30. There you need to select the instance then click on actions, now click on security, then click on modify IAM role.
31. Then select your policy, and click on update IAM role.





32. After all this is done go back to console and run the push command again.



33. After the first command has executed successfully, then you need to run the third command because the second command says to build the docker, but you have already built a docker. So, you do not require to use that command.
34. But before using the third command you need to change the name of the image. In your case the name of the image is **nginx** so, you need to change it to that.

```
[root@ip-172-31-21-113 ec2-user]# docker tag nginx:latest 463646775279.dkr.ecr.us-west-2.amazonaws.com/private-ecr-repository:latest
```

35. If you do a **docker images** commands you will be able to see your image.

```
[root@ip-172-31-21-113 ec2-user]# docker images
REPOSITORY                                                      TAG       IMAGE ID       CREATED         SIZE
nginx                                                           latest    d453dd892d93   2 months ago    187MB
463646775279.dkr.ecr.us-west-2.amazonaws.com/private-ecr-repository   latest    d453dd892d93   2 months ago    187MB
[root@ip-172-31-21-113 ec2-user]#
```

36. So, now you need to use the last command, that is docker push command. This command will push the docker image to the repository.

```
[root@ip-172-31-21-113 ec2-user]# docker push 463646775279.dkr.ecr.us-west-2.amazonaws.com/private-ecr-repository:latest
The push refers to repository [463646775279.dkr.ecr.us-west-2.amazonaws.com/private-ecr-repository]
b074db3b55e1: Pushed
e50c68532c4a: Pushed
f6ba584ca3ec: Pushed
01aaa195cdad: Pushed
2a13e6a7cca6: Pushed
370869eba6e9: Pushed
7292cf786aa8: Pushed
latest: digest: sha256:4669f6671aca20a34c3dfcd017e84fb3cae40788ea664866eaea698e3dfe241c size: 1778
[root@ip-172-31-21-113 ec2-user]#
```

37. Now, if you go back to your repository, you can see your image with tag of latest.