6. How do you implement persistent storage with Fargate?

Answer:

Fargate tasks are ephemeral by default, which means data is lost once the task stops. To persist data:

- I use **EFS (Elastic File System)** – a managed, scalable file system by AWS.
- Steps:
    1. Create an **EFS file system** in the same VPC.
    2. Mount it to the Fargate task using **task definition volumes**.
    3. Define the mount point inside the container so that any data written there persists.
- This is useful for shared or durable storage like logs, user uploads, or session data.

🔷 **Step 1: Create an EFS File System**

1. Go to the **Amazon EFS Console**.
2. Click **Create file system**.
3. Choose the **same VPC** as your Fargate service.
4. Under **Access points**, optionally create one (recommended for managed permissions).
5. Add mount targets in the **same subnets and AZs** where your Fargate tasks run.
6. Ensure your **EFS security group** allows inbound NFS traffic (port 2049) from Fargate security group.

🔷 **Step 2: Task Execution Role (IAM Permissions)**

Ensure the task execution role has the following policy:

json

```
{
 "Effect": "Allow",
 "Action": [
  "elasticfilesystem:ClientMount",
  "elasticfilesystem:ClientWrite",
  "elasticfilesystem:DescribeMountTargets"
 ],
 "Resource": "*"
```

}

🔷 **Step 3: Create ECS Task Definition with EFS Volume**

Here is a sample **Fargate ECS task definition (JSON)** with EFS:

json

```
{
 "family": "fargate-efs-demo",
 "networkMode": "awsvpc",
 "requiresCompatibilities": ["FARGATE"],
 "cpu": "256",
 "memory": "512",
 "executionRoleArn": "arn:aws:iam::<account-id>:role/ecsTaskExecutionRole",
 "containerDefinitions": [
  {
   "name": "app",
   "image": "nginx",
   "portMappings": [
    {
     "containerPort": 80,
     "protocol": "tcp"
    }
   ],
   "mountPoints": [
    {
     "sourceVolume": "efs-volume",
     "containerPath": "/usr/share/nginx/html",  // your mount path
     "readOnly": false
    }
   ],
   "logConfiguration": {
    "logDriver": "awslogs",
    "options": {
     "awslogs-group": "/ecs/efs-demo",
     "awslogs-region": "us-east-1",
     "awslogs-stream-prefix": "ecs"
    }
   }
  }
 ],
 "volumes": [
  {
   "name": "efs-volume",
   "efsVolumeConfiguration": {
    "fileSystemId": "fs-12345678",  // replace with your EFS ID
    "transitEncryption": "ENABLED",
    "authorizationConfig": {
```

```
      "accessPointId": "fsap-abcdefgh123456789",  // optional but recommended
      "iam": "ENABLED"
     }
    }
   }
 ]
}
```

🔷 **Step 4: Launch Fargate Task**

- Make sure you use:
  - Same **VPC** and **subnet** as EFS.
  - Correct **security groups** that allow NFS (port 2049) from ECS task ENIs.

🔷 **Step 5: Verify**

- Access the container and create/write files to the mounted directory.
- Stop and re-run the task. Files will persist.

---

7. Explain how networking works in Fargate tasks.

Answer:

Fargate uses **awsvpc mode** for networking. That means:

- Each task gets its own **ENI (Elastic Network Interface)** and **private IP** address.
- It behaves like a separate EC2 instance within the **VPC**.
- You can control traffic using:
  - **Subnets** (public/private),
  - **Security Groups** (like firewalls),
  - **Route tables** and **NACLs**.
- For internet access:
  - Public subnet + public IP, or
  - Private subnet + NAT Gateway.

This makes Fargate secure, as each task is isolated at the network level.

---

8. How do you troubleshoot Fargate tasks that fail to start?

Answer:

I follow this checklist to debug Fargate tasks:

1. **Check CloudWatch Logs**:
   - Look for stdout or stderr logs from the container.
2. **Describe Task**:
   - Run aws ecs describe-tasks to see error messages (e.g., RESOURCE:MEMORY, CannotPullContainerError).
3. **IAM Role Issues**:
   - Ensure the task role and execution role have proper permissions (e.g., to pull from ECR or write to CloudWatch).
4. **Image Pull Failures**:
   - Verify the image exists in ECR or Docker Hub and credentials are correct.
5. **Check Networking**:

- Make sure the security groups and subnets allow necessary communication (e.g., DB ports).

---

9. What are the best practices for optimizing Fargate costs?

Answer:

To reduce Fargate costs:

- **Right-size tasks**: Avoid over-provisioning CPU/memory.
- **Use Spot Fargate** for non-critical workloads (~70% cheaper).
- **Task Auto Scaling**: Use ECS Service Auto Scaling to run only what you need.
- **Use Graviton-based compute**: If supported, they're cheaper and more efficient.
- **Shut down idle tasks**: Schedule cleanup of unused resources.
- **Monitor usage**: Set CloudWatch alarms for unexpected cost spikes.

---

10. How do you implement logging and monitoring for Fargate tasks?

Answer:

Logging:

- Enable **CloudWatch logging** in task definition:
  - Add a **logConfiguration** block to use the awslogs driver.
  - Each task sends logs to a specific log group.

Monitoring:

- Use **CloudWatch metrics** to track:
  - CPU, memory usage, task health.
- Use **CloudWatch Alarms** to alert on high usage or failures.
- For more insights:
  - Integrate with **AWS X-Ray** or **Datadog/Grafana/Prometheus** if needed.

---

11. Explain how to use Fargate with VPC endpoints for private connectivity.

Answer:

If I want my Fargate task to securely access AWS services **without using the public internet**, I use **VPC Endpoints**:

- **Step-by-step**:
  1. Create a **VPC Endpoint** (e.g., for S3, DynamoDB, or ECR).
  2. Make sure Fargate tasks run in **private subnets**.
  3. Attach proper **IAM roles** and **route tables**.
  4. The task now communicates directly over AWS's internal network – **no need for NAT Gateway or internet gateway**.

This is important for **secure, low-latency access** to AWS services in private environments.

Deploy an NGINX Docker App on AWS Fargate

Step 1: Create a Docker Image & Push to AWS ECR

🔧 **Goal:**

Package your app into a Docker image and upload it to AWS Elastic Container Registry (ECR).

👣 **Steps:**

1. **Create a folder for your app:**

bash
---
mkdir fargate-demo && cd fargate-demo

2. **Add a basic Dockerfile using NGINX:**

bash
---
echo "FROM nginx:alpine" > Dockerfile

3. **Build the image:**

bash
---
docker build -t fargate-demo .

4. **Create an ECR repository in AWS to store this image:**

bash
---
aws ecr create-repository --repository-name fargate-demo
✅ This will return output like:
json

"repositoryUri": "906253564515.dkr.ecr.us-east-2.amazonaws.com/fargate-demo"

5. **Tag the Docker image to point to your ECR:**

bash
---
aws_account_id=$(aws sts get-caller-identity --query Account --output text)
region=$(aws configure get region)
docker tag fargate-demo:latest
"$aws_account_id.dkr.ecr.$region.amazonaws.com/fargate-demo:latest"

6. **Login to ECR and push the image:**

bash
---
aws ecr get-login-password | docker login --username AWS --password-stdin
"$aws_account_id.dkr.ecr.$region.amazonaws.com"
docker push "$aws_account_id.dkr.ecr.$region.amazonaws.com/fargate-demo:latest"

🛠️ Step 2: Create ECS Cluster

🔧 **Goal:**
Create a place where your app will run inside ECS.
👣 **Command:**
bash
---
aws ecs create-cluster --cluster-name fargate-demo-cluster
✅ Output will show your cluster details:
json

"clusterName": "fargate-demo-cluster", "status": "ACTIVE"

Step 3: Define Task Definition

🔧 **Goal:**

Tell ECS **how to run** your container (CPU, memory, image, port, etc.)
👣 **Create a file called task-definition.json with:**
json

```
{
 "family": "fargate-demo-task",
 "requiresCompatibilities": ["FARGATE"],
 "networkMode": "awsvpc",
 "cpu": "256",
 "memory": "512",
 "executionRoleArn": "arn:aws:iam::<ACCOUNT_ID>:role/ecsTaskExecutionRole",
 "containerDefinitions": [
  {
   "name": "fargate-demo-container",
   "image": "<ACCOUNT_ID>.dkr.ecr.<REGION>.amazonaws.com/fargate-demo:latest",
   "portMappings": [
    {
     "containerPort": 80,
     "protocol": "tcp"
    }
   ],
   "essential": true
  }
 ]
}
```
Replace <ACCOUNT_ID> and <REGION> with your actual values.

Step 4: Register Task Definition
Goal:
Let ECS know about the task definition you created.
👣 **Command:**
bash

```
aws ecs register-task-definition --cli-input-json file://task-definition.json
```
✅ Output shows successful registration:
json

```
"taskDefinitionArn": "arn:aws:ecs:us-east-2:906253564515:task-definition/fargate-demo-task:1"
```

Step 5: Create Security Group and Run the Task
🔧 **Goal:**
Allow web access and run the container.
👣 **Steps:**
1. **Get default VPC and subnet:**

bash

```
vpc_id=$(aws ec2 describe-vpcs --filters Name=isDefault,Values=true --query "Vpcs[0].VpcId" --output text)
subnet_id=$(aws ec2 describe-subnets --filters Name=vpc-id,Values=$vpc_id --query "Subnets[0].SubnetId" --output text)
```

2. **Create security group and allow port 80 (HTTP):**

```
bash
-----
sg_id=$(aws ec2 create-security-group --group-name fargate-sg --description "Fargate SG" --vpc-id $vpc_id --query "GroupId" --output text)

aws ec2 authorize-security-group-ingress --group-id $sg_id --protocol tcp --port 80 --cidr 0.0.0.0/0
```

3. **Run the Fargate task:**

```
bash
-----
aws ecs run-task \
  --cluster fargate-demo-cluster \
  --launch-type FARGATE \
  --network-configuration "awsvpcConfiguration={subnets=[$subnet_id],securityGroups=[$sg_id],assignPublicIp=ENABLED}" \
  --task-definition fargate-demo-task
```

Step 6: Access Your App in Browser

🔧 **Goal:**

Get the **public IP** of the running container and open it in a browser.

👣 **Steps:**

1. **Get the running task ARN:**

```
bash
-----
task_arn=$(aws ecs list-tasks --cluster fargate-demo-cluster --query "taskArns[0]" --output text)
```

2. **Get the network interface (ENI):**

```
bash
```

```
eni_id=$(aws ecs describe-tasks --cluster fargate-demo-cluster --tasks $task_arn --query "tasks[0].attachments[0].details[?name=='networkInterfaceId'].value" --output text)
```

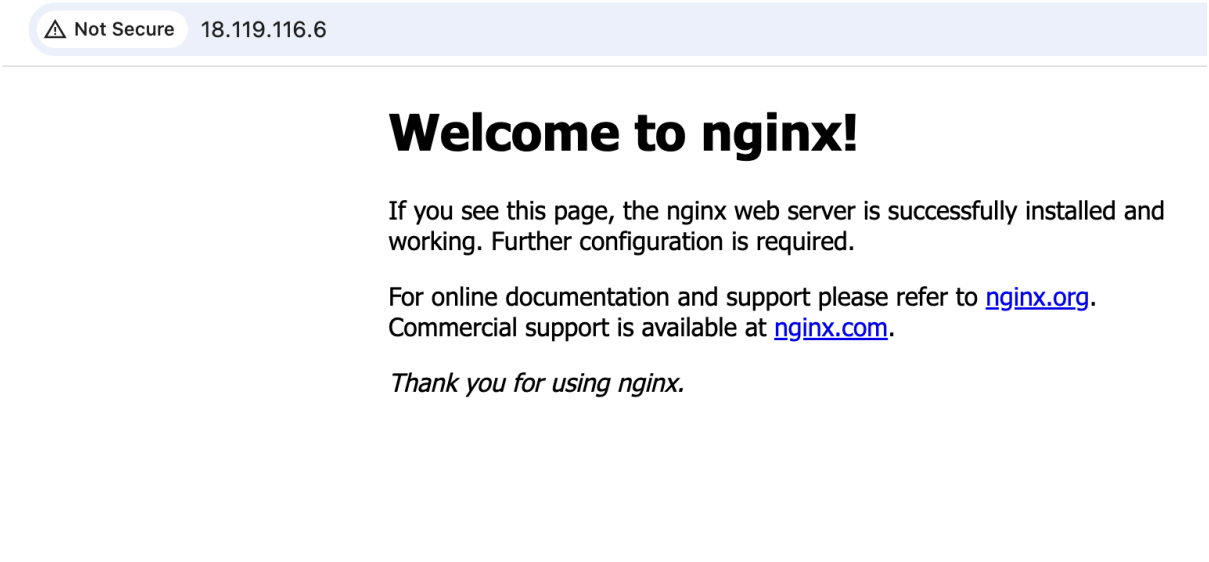
3. **Get the public IP:**

```
bash
-----
aws ec2 describe-network-interfaces --network-interface-ids $eni_id --query "NetworkInterfaces[0].Association.PublicIp" --output text
```

4. **Open in browser:**

```
cpp
-----
```

http://<public-ip>
You'll see the NGINX welcome page!



How can you implement persistent storage for a Fargate task?
A. Attach EBS volumes directly to the Fargate task
B. Use S3 to mount as a filesystem
C. Use EFS and mount it via task definition
D. Enable data persistence in task settings

✅ **Correct Answer:** C. Use EFS and mount it via task definition

---

Q. Which networking mode is used by AWS Fargate for tasks?
A. bridge
B. host
C. awsvpc
D. overlay

✅ **Correct Answer:** C. awsvpc