Best Security and Production-Readiness Practices for AWS ECS Fargate with PHPFPM, Nginx, and RDS Aurora (2024)

1. Network Security for ECS Fargate

a. Virtual Private Cloud (VPC) Design and Subnet Configuration

• Private Subnets for ECS Tasks:

o Always place your ECS Fargate tasks in private subnets to prevent

direct access from the public internet.

o Ensure outbound traffic is routed through a NAT Gateway for internet

access.

o Separate application layers (e.g., PHP-FPM, Nginx) and databases

into distinct subnets to create a segmented, layered architecture that

enhances security.

Steps to configure:

o In the VPC Console, create a new VPC (or modify an existing one) and

define both public and private subnets in multiple availability zones

(AZs).

o Set up a NAT Gateway for the private subnets to ensure ECS tasks can

reach the internet for updates or external API calls.

b. Security Groups and Network ACLs

• Security Groups should follow the principle of least privilege, where only

required ports are open:

o Application Load Balancer (ALB): Allow traffic on ports 80 (HTTP) and

443 (HTTPS).

o PHP-FPM containers: Allow only traffic from Nginx (e.g., on port

9000).

o RDS Aurora: Restrict access to your DB security group, allowing only

traffic from ECS tasks or jump servers via port 3306 (MySQL).

o Implement Network ACLs as a second layer of defense, ensuring that

only necessary traffic traverses the subnet level.

Configuration:

o Security Groups: In the EC2 Console, create or modify Security

Groups to tightly restrict traffic.

o Network ACLs: Use the VPC Console to implement stateless network

traffic rules at the subnet level, further restricting access.

c. VPC Flow Logs for Monitoring

• Enable VPC Flow Logs to capture IP traffic data to and from your VPC. This

is critical for identifying potential intrusions or unusual traffic patterns.

o Log data can be stored in S3 or streamed directly to CloudWatch Logs

for monitoring.

2. IAM Roles and Policies

a. ECS Task Execution Role

• Create an IAM Role for ECS tasks with minimal permissions necessary for

execution:

o AmazonECSTaskExecutionRolePolicy should be attached to allow ECS

tasks to pull container images from ECR and send logs to

CloudWatch.

o IAM Policies for accessing AWS Secrets Manager, S3, and other

necessary services should be included as custom policies, granting

only the necessary permissions for that specific ECS service.

Configuration Steps:

o In the IAM Console, create a role for ECS tasks and attach only the

necessary policies, such as AmazonECSTaskExecutionRolePolicy and

any custom policies for secret management or logging.

b. RDS Aurora Database Access

• Use IAM roles to control database access, eliminating hardcoded credentials

in container images.

o For PHP applications, use the AWS SDK to retrieve credentials from

Secrets Manager or Parameter Store.

Sample IAM policy for Secrets Manager:

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": "secretsmanager:GetSecretValue",

"Resource": "arn:aws:secretsmanager:region:account-id:secret:your-secretname"

}

]

}

3. Container Security and Hardening

a. Minimal Docker Base Images

• Use slim or Alpine-based images to reduce the attack surface of your

containers.

o Use the PHP-FPM Alpine image and the Nginx Alpine image to keep

your containers lightweight and secure:

o FROM php:8.1-fpm-alpine

o FROM nginx:alpine

b. Non-Root User for Container Processes

• Always run containerized applications as non-root users to limit the

privileges of container processes:

o In your Dockerfile, add:

o USER nginx

o For PHP-FPM, ensure that the container doesn’t have elevated

privileges unless strictly necessary.

c. ECS Task Definition Privilege Restrictions

• In the ECS Task Definition, explicitly set "privileged": false to prevent

containers from gaining extra privileges on the host.

Sample ECS Task Definition Snippet:

{

"containerDefinitions": [

{

"name": "php-fpm",

"privileged": false,

"logConfiguration": { ... }

}

]

}

d. ECR Image Scanning

• Enable image scanning in Amazon ECR to automatically scan your container

images for vulnerabilities.

o Enable Enhanced Scanning for deeper analysis of security issues in

third-party libraries and dependencies.

4. Secrets Management

a. Using AWS Secrets Manager

• Store sensitive information like database credentials, API keys, and tokens in

AWS Secrets Manager. Avoid storing secrets in environment variables or

hardcoding them into container images.

o Secrets can be fetched at runtime using the AWS SDK or directly

referenced in the ECS task definition using the valueFrom parameter.

Task Definition Example for Secrets Manager Integration:

{

"name": "DB\_PASSWORD",

"valueFrom": "arn:aws:secretsmanager:region:account-id:secret:your-secret-id"

}

b. Rotate Secrets Regularly

• Use AWS Secrets Manager to enable automatic secret rotation to enhance

security. For example, enable RDS Aurora password rotation and ensure

that your application is configured to use the updated credentials.

5. TLS Encryption for In-Transit Data

a. TLS for Nginx Application

• Use AWS ACM (Certificate Manager) to manage SSL certificates and apply

TLS encryption for traffic to your Application Load Balancer (ALB).

o Always enforce HTTPS connections and redirect HTTP traffic to HTTPS

at the ALB or Nginx level.

Nginx Configuration Example for Redirect:

server {

listen 80;

server\_name example.com;

return 301 https://$host$request\_uri;

}

b. Database SSL Enforcement

• Enforce SSL encryption for connections between your ECS containers and

RDS Aurora. Use the Amazon RDS SSL certificates and require SSL in the

PDO or MySQLi connection string in PHP.

PHP Example for Enforcing SSL:

$pdo = new PDO(

'mysql:host=your-db-host;dbname=your-db-name',

$username,

$password,

array(PDO::MYSQL\_ATTR\_SSL\_CA => '/path/to/rds-combined-ca-bundle.pem')

);

6. Monitoring, Logging, and Performance Insights

a. CloudWatch Logs for ECS

• Configure CloudWatch Logs for ECS to monitor your application and

container logs. Ensure that logs from both Nginx and PHP-FPM are being

sent to CloudWatch for central logging.

Sample ECS Task Definition for CloudWatch Logs:

"logConfiguration": {

"logDriver": "awslogs",

"options": {

"awslogs-group": "/ecs/app-logs",

"awslogs-region": "us-east-1",

"awslogs-stream-prefix": "ecs"

}

}

b. Enhanced Monitoring and Performance Insights for RDS Aurora

• Enable Enhanced Monitoring and Performance Insights on your RDS

Aurora cluster to gain detailed metrics and visibility into query

performance, resource usage, and bottlenecks.

To enable Performance Insights:

o Go to RDS Console > Modify Cluster > Enable Performance Insights

and Enhanced Monitoring.

o Use CloudWatch Dashboards to visualize database performance.

c. CloudWatch Alarms for ECS and RDS

• Set up CloudWatch Alarms on key metrics such as CPU, memory usage, and

error rates for ECS containers, and latency, throughput, and query

execution time for RDS Aurora.

o For ECS, set alarms on ECS service metrics like CPUUtilization,

MemoryUtilization, and ALB request count.

o For

RDS, monitor DB connections, read/write latency, and replication lag.

7. Backup and Disaster Recovery

a. Automated Backups and Snapshots for RDS

• Enable automated backups for your RDS Aurora cluster with a sufficient

retention period (e.g., 7-14 days).

o Regularly schedule manual snapshots for important production

releases or before major changes.

b. Point-in-Time Recovery

• Point-in-Time Recovery allows you to restore your database to any point in

time within the retention period. This ensures that even in the case of

human error or corruption, you can restore the DB to a consistent state.

8. Auto Scaling and Load Balancing

a. ECS Auto Scaling Policies

• Enable Auto Scaling for ECS tasks to automatically adjust the number of

running tasks based on CPU or memory utilization. Configure scaling

policies based on key performance indicators to ensure application

availability during traffic spikes.

How to configure ECS Auto Scaling:

o In the ECS Console, go to your service, and under the Auto Scaling

tab, configure policies for scaling in and out based on CloudWatch

metrics.

b. RDS Aurora Auto Scaling for Read Replicas

• Enable Aurora Auto Scaling to dynamically scale the number of read

replicas in response to read traffic patterns.

Configuration Steps:

o In the RDS Console, go to your Aurora cluster, and configure Auto

Scaling for the Reader Endpoint to add or remove replicas based on

CPU utilization or throughput.

9. Web Application Firewall (WAF) and DDoS Protection

a. AWS WAF Integration

• Deploy AWS WAF on your Application Load Balancer to filter malicious

traffic and block attacks such as SQL Injection, Cross-Site Scripting (XSS),

and bots.

o Create a Web ACL in the WAF Console and apply rules for common

attack patterns.

b. AWS Shield (Standard and Advanced)

• Enable AWS Shield Standard for DDoS protection, included by default with

all AWS services.

o For critical applications, consider AWS Shield Advanced for enhanced

DDoS protection and 24/7 incident response support.

Conclusion

By following these best practices, your ECS Fargate and RDS Aurora setup will be

highly secure, scalable, and resilient for production environments. Regularly

review and refine your security posture, logging configurations, and scaling

policies to align with evolving requirements and AWS updates.