

Project Title: Loan Management with AWS

Introduction:

In today's fast-paced financial landscape, managing loan processes efficiently and securely is more critical than ever. Our project, "Loan Management with AWS," revolutionizes how financial institutions handle loans, from application to repayment, by harnessing the power of cloud technology. Utilizing a robust framework powered by Amazon Web Services (AWS), our system offers a seamless, scalable, and secure platform for both loan officers and customers.

The core of our innovation lies in our deployment on AWS EC2, backed by a MongoDB database that ensures rapid processing and high availability. We have meticulously designed a custom Virtual Private Cloud (VPC) to safeguard all transactions with top-tier security measures, including AWS Security Groups and AWS Key Management Service (KMS) for stringent data encryption. Our application, developed using the flexible and dynamic Flask framework, supports rapid development and easy integration, making it adaptable to varying business needs and scalable to accommodate growing user numbers.

Our system addresses common issues faced by startups and smaller platforms, such as limited resources that lead to performance bottlenecks, insufficient security measures, and lack of comprehensive loan management features. By leveraging AWS technologies like Elastic Container Service (ECS) and Elastic Container Registry (ECR), our solution scales efficiently, managing spikes in web traffic gracefully without compromising on performance.

Moreover, we employ AWS WAF and extensive monitoring via AWS CloudWatch to protect against and swiftly respond to potential cyber threats, ensuring continuous operation without service interruptions. With these integrated AWS services, It not only enhances operational efficiency but also elevates user experience by providing a robust, intuitive, and accessible platform for managing loans effectively.

Our loan management system exemplifies cutting-edge technology's role in transforming financial services, offering unparalleled reliability, security, and user-friendly features that empower both lenders and borrowers in the digital age.

Issues with existing platforms

Existing loan management platforms, especially those developed by startups with limited resources, often face significant challenges impacting their performance and reliability. These issues include:

- **Scalability and Performance:** Many existing systems struggle to handle large volumes of users or data efficiently. This can lead to slow application response times and, in some cases, Application crashes when too many users access the system simultaneously. These performance issues are particularly problematic during peak operational times, which can deter user satisfaction and trust.
- **Limited Loan Management Features:** Due to constrained development capabilities, some platforms may not offer a broad range of loan options or customization features. This lack of versatility can hinder the ability to meet diverse customer needs and preferences, which is essential in the competitive financial services market.

- **Inadequate Security Measures:** Security is a paramount concern in any financial application, yet many startups struggle to implement robust security protocols. This can leave sensitive financial information vulnerable to breaches. Common shortcomings include insufficient data encryption and lack of comprehensive security practices to protect user data during transactions.
- **Poor Integration of Advanced Analytical Tools:** Many platforms do not incorporate advanced tools for analytics and reporting, which are crucial for users managing their loans and for officers assessing loan applications. The absence of these tools can affect the decision-making process and operational efficiency.
- **Infrastructure and Maintenance Costs:** Maintaining and scaling traditional loan management systems can be costly and complex, requiring significant IT expertise and resources. This can be a barrier for smaller institutions looking to expand their services or improve system reliability.

What and How

Application Development Using Flask

What We Did: We chose Flask, a lightweight and flexible Python web framework, for developing our application. Flask's simplicity and capability for rapid development make it ideal for our needs, allowing for easy integration with other services and a modular approach to building applications.

How We Did It: Flask facilitates the creation of modular and maintainable code. This is crucial for adapting quickly to changing business requirements or scaling up the application features. It supports integration with a variety of AWS services, enhancing our application's functionality and scalability.

Database Management with MongoDB on AWS EC2

What We Did: We used MongoDB, a NoSQL database, because of its scalability and flexibility in handling large datasets and complex queries. Hosting it on AWS EC2 allowed us to leverage the robustness and manageability of AWS infrastructure.

How We Did It: AWS EC2 is fully compatible with MongoDB applications, which enabled us to use existing MongoDB tools and skills effectively. This setup provides the necessary durability, scalability, and security needed for enterprise applications dealing with sensitive financial data.

Application Scalability with ECS and AWS Load Balancer

What We Did: To ensure that our application could scale effectively and handle varying loads, we deployed it on AWS ECS (Elastic Container Service). This service manages containerized applications and automates the scaling and management of applications.

How We Did It: ECS handles our Docker containers and adjusts the number of active instances according to the load. This auto-scaling feature is crucial for maintaining performance during peak usage times. The AWS Load Balancer complements this by distributing incoming traffic across these instances, further ensuring consistent and smooth application performance.

Security with WAF and AWS Security Groups

What We Did: We fortified our application with AWS WAF (Web Application Firewall) and AWS Security Groups. These tools provide robust security measures to protect against unauthorized access and web-based threats.

How We Did It: AWS WAF: Protects our application from common web exploits and attacks by allowing us to define customizable web security rules.

Security Groups: These act as virtual firewalls for our AWS services, controlling both inbound and outbound traffic. This ensures that only authorized traffic can access our application, providing an additional layer of security.

Loan management Features

User Submissions

Property Details: Users can upload comprehensive property information, including type, location, valuation, and ownership documentation, which is crucial for securing mortgage loans.

Bank Details: Users provide essential bank information, such as bank name, account number, and IFSC code, ensuring seamless transactions and verifications.

Personal Details: Includes critical personal information such as full name, date of birth, income, and contact info, which helps in assessing the creditworthiness of the applicant.

Loan Request & Approval

Loan Amount: Users can specify the desired borrowing amount, tailoring the loan to meet their specific financial needs.

Officer Review: A loan officer reviews each application, assessing it based on risk, borrower's credit history, and compliance with lending criteria. The officer then either approves or rejects the loan request.

Post-Approval Management

EMI Payments: After loan approval, users can set up and make monthly installments (EMIs) directly through the platform. This feature simplifies the repayment process and helps users manage their debts effectively.

Loan Tracking: Users have access to a dashboard where they can monitor the status of their loan, view payment history, and check the remaining balance. This feature provides transparency and allows users to keep track of their financial obligations in real-time.

Aws Services used

AWS EC2 MongoDB:

- **Implementation:** We utilized Amazon EC2 instances to deploy MongoDB, configuring these instances to handle our database requirements efficiently. The setup ensured that our MongoDB database was scalable and capable of managing the increasing data volume as user numbers grew.

AWS ECR (Elastic Container Registry):

- **Implementation:** We began by pushing our Docker images, which include the application environment setup and dependencies, to AWS ECR. This provided a reliable and secure repository for our container images, simplifying version control and deployment processes.

AWS ECS (Elastic Container Service):

- **Implementation:** After storing our images in ECR, we used AWS ECS to manage our Docker containers. We set up ECS services and task definitions that dictate how containers should run—

such as CPU and memory usage, scaling requirements, and networking settings. This allowed our application to automatically scale based on demand, ensuring high availability and fault tolerance.

AWS Secrets Manager:

- **Implementation:** To manage credentials and other secrets needed by our application, we utilized AWS Secrets Manager. This service helped us replace hard-coded credentials in our code with calls to Secrets Manager, which significantly reduced the risk of security leaks.

AWS WAF (Web Application Firewall):

- **Implementation:** We deployed AWS WAF to protect our web application from common exploits and attacks by defining customizable web security rules. This setup blocked malicious traffic and prevented data breaches, ensuring the integrity and security of our application interfaces.

AWS Load Balancer:

- **Implementation:** We configured an AWS Load Balancer to distribute incoming application traffic across multiple EC2 instances running our application. This not only balanced the load effectively but also improved the application's overall performance and resilience to potential failures.

AWS KMS (Key Management Service):

- **Implementation:** We integrated AWS KMS to manage encryption keys for our sensitive data, such as user personal information and financial details. KMS provided us with the tools to create and control encryption keys, enhancing our application's data security by ensuring that encryption practices met the highest standards.

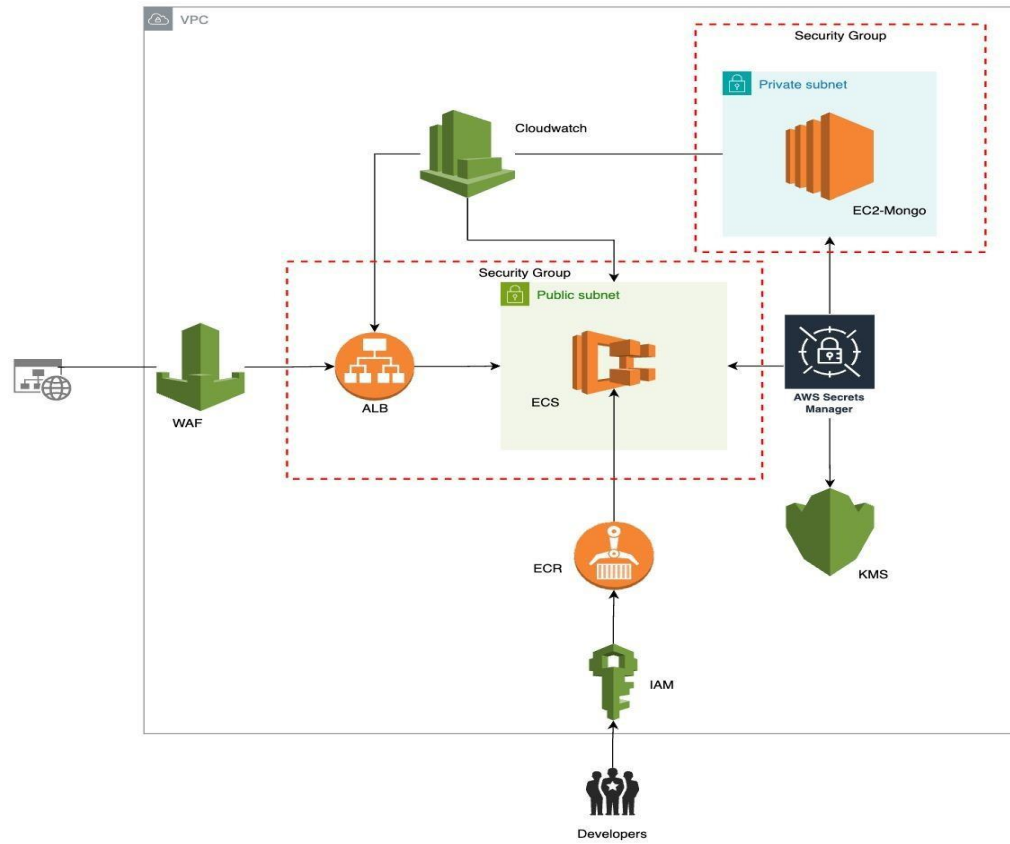
AWS Security Groups:

- **Implementation:** We configured Security Groups in AWS as a virtual firewall for our EC2 instances to strictly control both incoming and outgoing traffic. This setup ensured that only traffic according to the defined security rules could access our services, safeguarding our infrastructure from unauthorized access.

AWS IAM (Identity and Access Management):

- **Implementation:** Lastly, we set up AWS IAM to manage permissions for users and systems that interact with our AWS resources. By defining roles and policies, IAM ensured that only authorized personnel and systems had access to specific AWS resources, enforcing a secure environment.

Application Architecture:



Creation of Services Step by Step

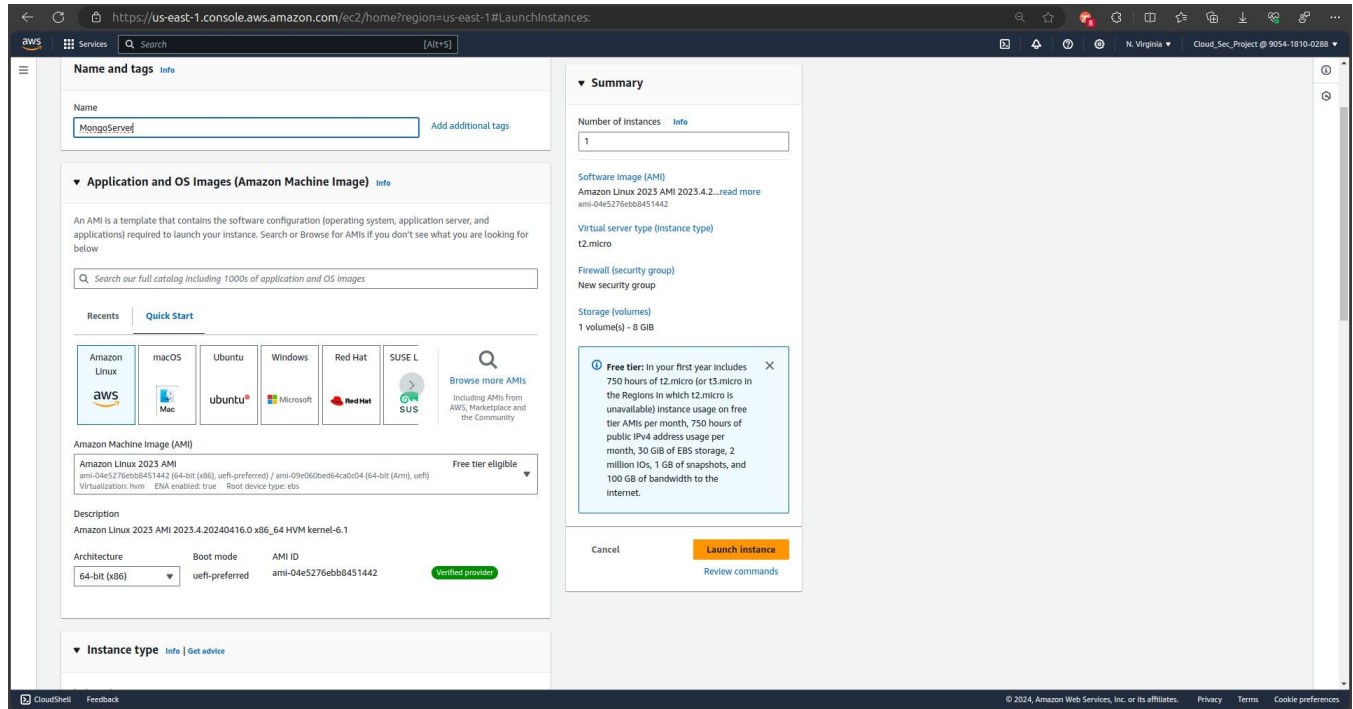
1. Creating an EC2 Instance for MongoDB Hosting

Step 1: Log in to your AWS Management Console.

Step 2: Navigate to the EC2 Dashboard and click on "Instances."

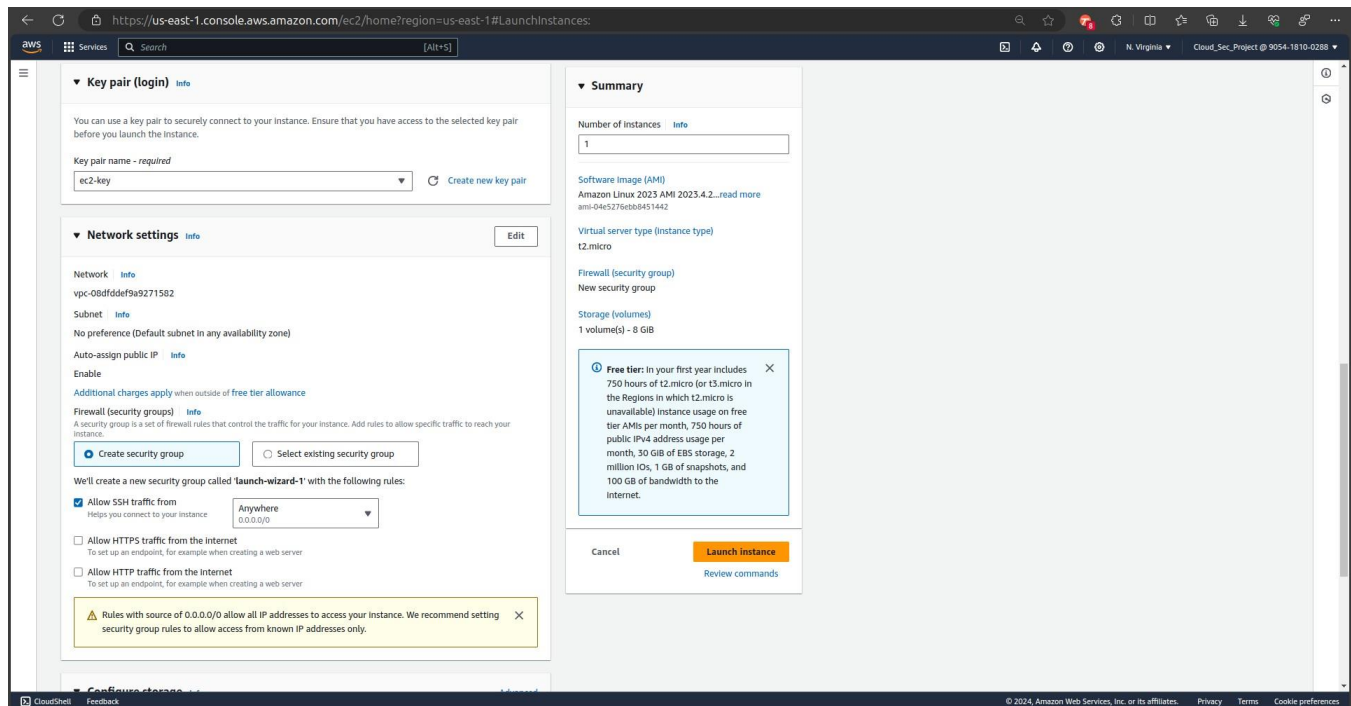
Step 3: Click on "Launch Instances."

Step 4: Select the Default Amazon Machine Image (AMI) that supports MongoDB.



Step 5: Select an instance type that meets our performance requirements.

Step 6: Configure instance details such as network, subnet, IAM role (if necessary), and monitoring.



Step 7: Configure security group rules to allow traffic on required ports (e.g., port 27017 for MongoDB).

Step 8: Review and launch the instance. Set up MongoDB after the instance is running.

2. Setup MongoDB in the EC2 Instance

Step 1: Create `/etc/yum.repos.d/mongodb-org-7.0.repo` file

Step 2:

```
[mongodb-org-7.0]
name=MongoDB Repository
baseurl=https://repo.mongodb.org/yum/amazon/2023/mongodb-org/7.0/x86_64/
gpgcheck=1
enabled=1
gpgkey=https://pgp.mongodb.com/server-7.0.asc
```

Add the above to the file created in step1 repo file.

Step 3: `sudo yum install -y mongodb-org`

Step 4: Edit the `/etc/mongod.conf` and set the bindIP: `0.0.0.0/0` to allow all hosts.

Step 5: `sudo systemctl restart mongod`

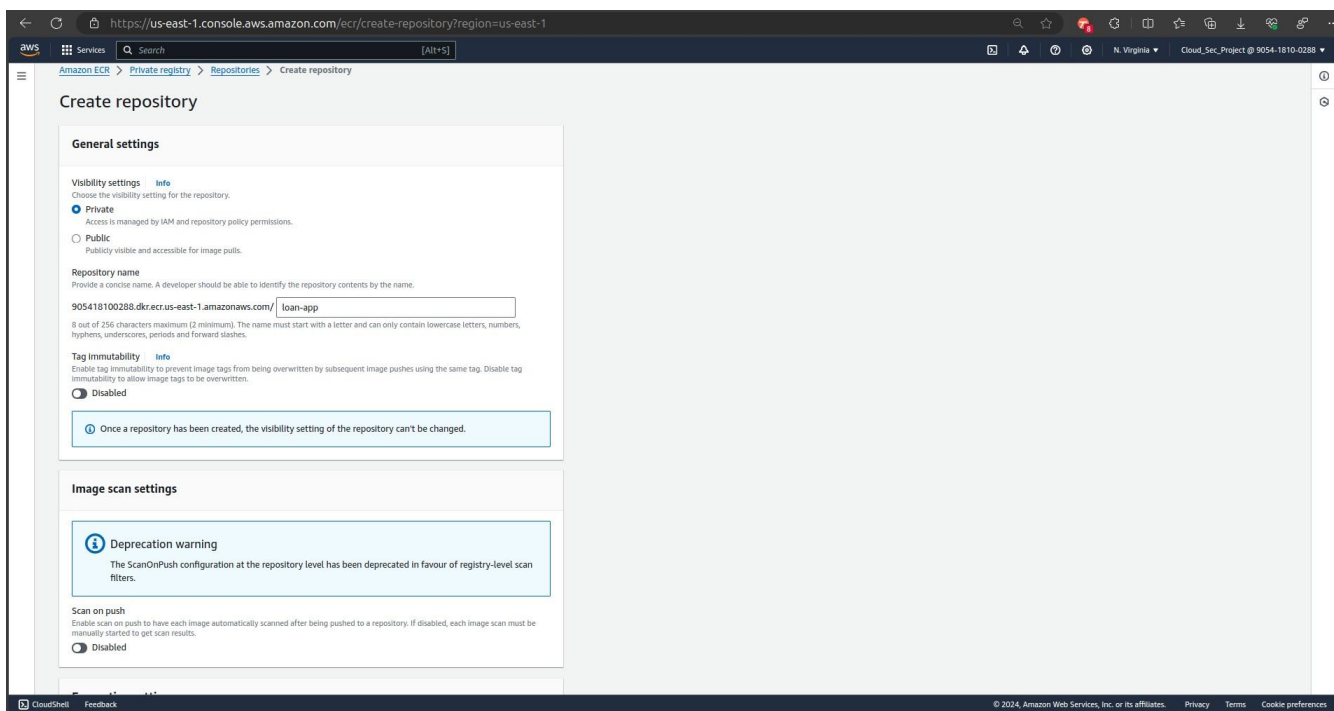
Step 6: `sudo systemctl enable mongod`

3. Creating ECR for Application Image

Step 1: Go to the Elastic Container Registry (ECR) service in the AWS Console.

Step 2: Click on "Repositories" then "Create repository."

Step 3: Enter a name for your repository.



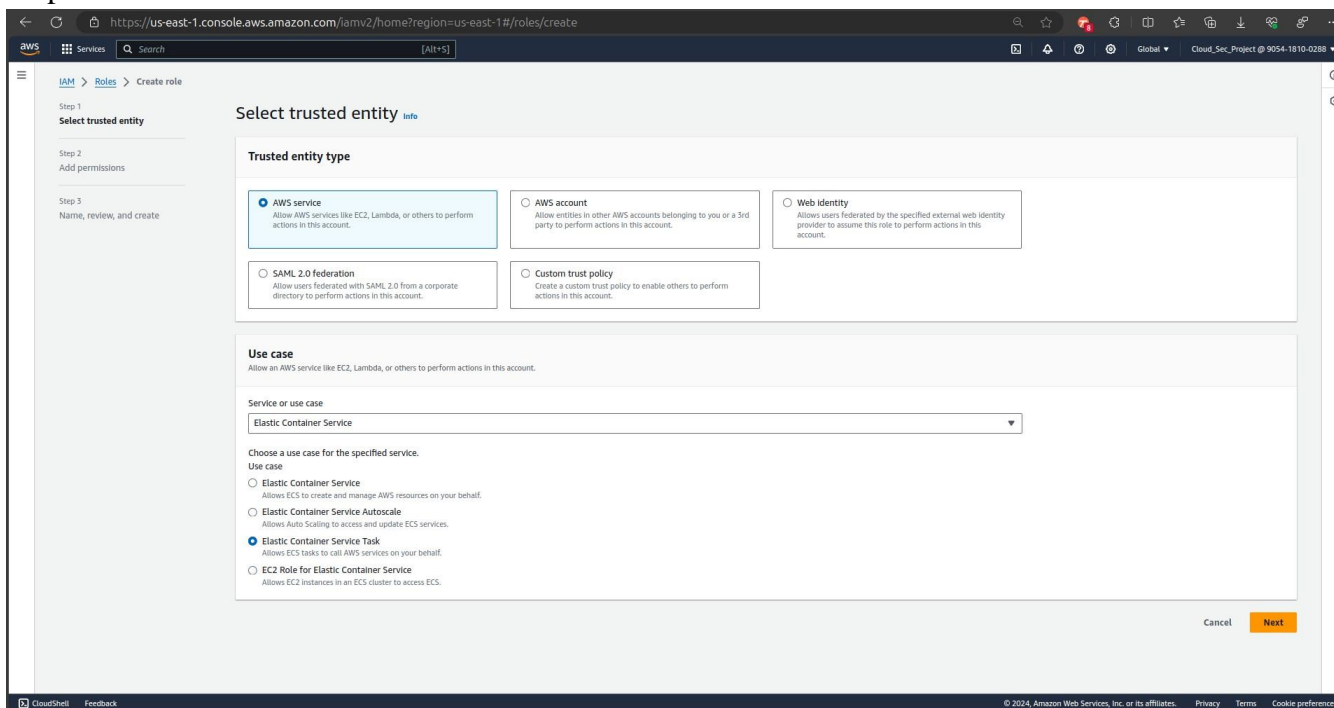
Step 4: Click "Create." You'll then use this repository to push and pull Docker images of your application.

4. Creating IAM Role for ECS Task

Step 1: Go to the IAM dashboard.

Step 2: Click on "Roles" then "Create role."

Step 3: Choose "ECS" as the service that will use this role.



Step 4: Attach policies that your ECS tasks need (e.g., access to ECR, logging, network configuration).

←

↻

https://us-east-1.console.aws.amazon.com/kms/home?region=us-east-1#/kms/keys/create

🔍 ☆ 📌 📄 📁 📂 📅 📆 📇 📈 📉 📊 📋 📌 📍 📎 📏 📐 📑 📒 📓 📔 📕 📖 📗 📘 📙 📚 📛 📜 📝 📞 📟 📠 📡 📢 📣 📤 📥 📦 📧 📨 📩 📪 📫 📬 📭 📮 📯 📰 📱 📲 📳 📴 📵 📶 📷 📸 📹 📺 📻 📼 📽 📾 📿 📰 📱 📲 📳 📴 📵 📶 📷 📸 📹 📺 📻 📼 📽 📾 📿

Services 🔍 Search [Alt+S]

📍 N. Virginia Cloud_Sec_Project @ 9054-1810-0288

KMS > Customer-managed keys > Create key

Step 1
Configure key

Step 2
Add labels

Step 3
Define key administrative permissions

Step 4
Define key usage permissions

Step 5
Review

Add labels

Alias

You can change the alias at any time. [Learn more](#)

Alias

custom-key

Description - optional

You can change the description at any time.

Description

Description of the key

Tags - optional

You can use tags to categorise and identify your KMS keys and help you track your AWS costs. When you add tags to AWS resources, AWS generates a cost allocation report for each tag. [Learn more](#)

This key has no tags.

Add tag

You can add up to 50 more tags.

Cancel

Previous

Next

←

↻

https://us-east-1.console.aws.amazon.com/kms/home?region=us-east-1#/kms/keys/create

🔍 ☆ 📌 📄 📁 📂 📅 📆 📇 📈 📉 📊 📋 📌 📍 📎 📏 📐 📑 📒 📓 📔 📕 📖 📗 📘 📙 📚 📛 📜 📝 📞 📟 📠 📡 📢 📣 📤 📥 📦 📧 📨 📩 📪 📫 📬 📭 📮 📯 📰 📱 📲 📳 📴 📵 📶 📷 📸 📹 📺 📻 📼 📽 📾 📿

Services 🔍 Search [Alt+S]

📍 N. Virginia Cloud_Sec_Project @ 9054-1810-0288

KMS > Customer-managed keys > Create key

Step 1
Configure key

Step 2
Add labels

Step 3
Define key administrative permissions

Step 4
Define key usage permissions

Step 5
Review

Define key administrative permissions

Key administrators (9/9)

Choose the IAM users and roles who can administer this key through the KMS API. You may need to add additional permissions for the users or roles to administer this key from this console. [Learn more](#)

Search Key administrators

< 1 >

<input checked="" type="checkbox"/>	Name	Path	Type
<input checked="" type="checkbox"/>	admin	/	User
<input checked="" type="checkbox"/>	Cloud_Sec_Project	/	User
<input checked="" type="checkbox"/>	AWSServiceRoleForConfig	/aws-service-role/config.amaz...	Role
<input checked="" type="checkbox"/>	AWSServiceRoleForCostOptim...	/aws-service-role/cost-optimiz...	Role
<input checked="" type="checkbox"/>	AWSServiceRoleForECS	/aws-service-role/ecs.amazon...	Role
<input checked="" type="checkbox"/>	AWSServiceRoleForElasticLoa...	/aws-service-role/elasticloadb...	Role
<input checked="" type="checkbox"/>	AWSServiceRoleForSupport	/aws-service-role/support.am...	Role
<input checked="" type="checkbox"/>	AWSServiceRoleForTrustedAd...	/aws-service-role/trustedadv...	Role
<input checked="" type="checkbox"/>	ecsTaskExecutionRole	/	Role

Key deletion

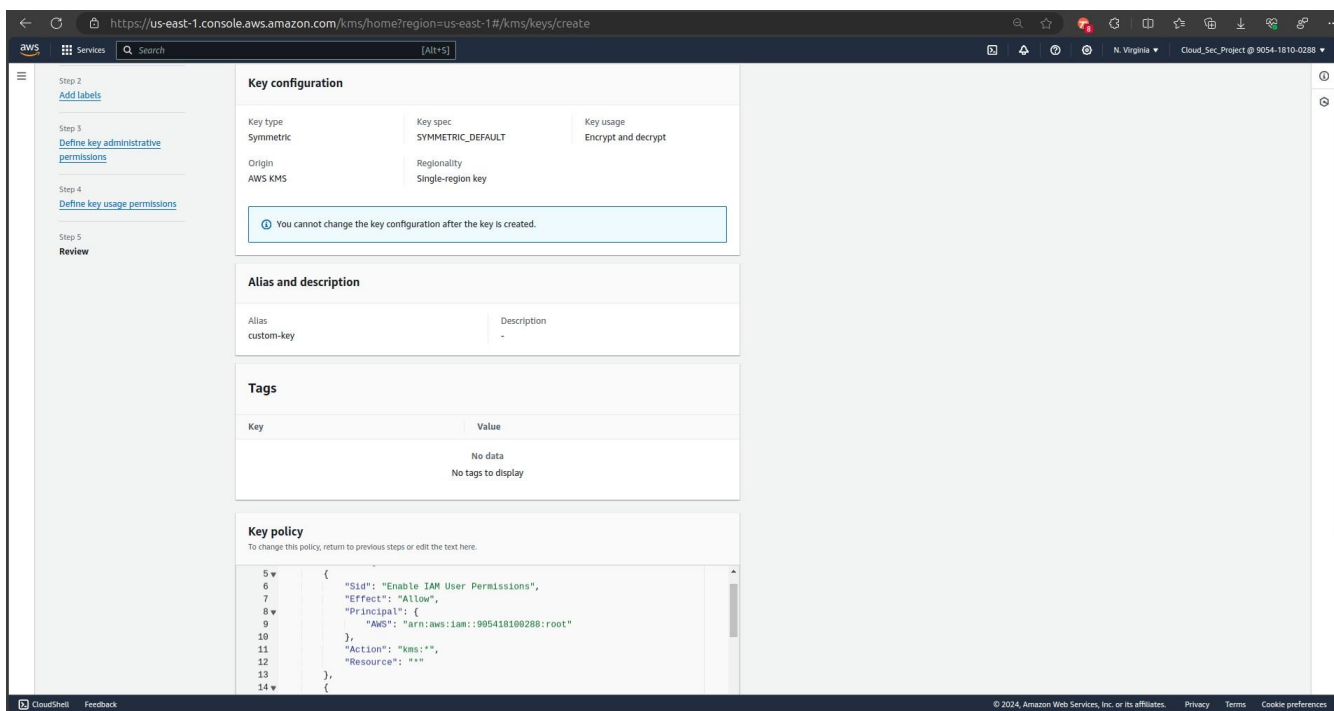
☒ Allow key administrators to delete this key.

Cancel

Previous

Next

Step 3: Complete the key creation process and use this key for encrypting data in other services.



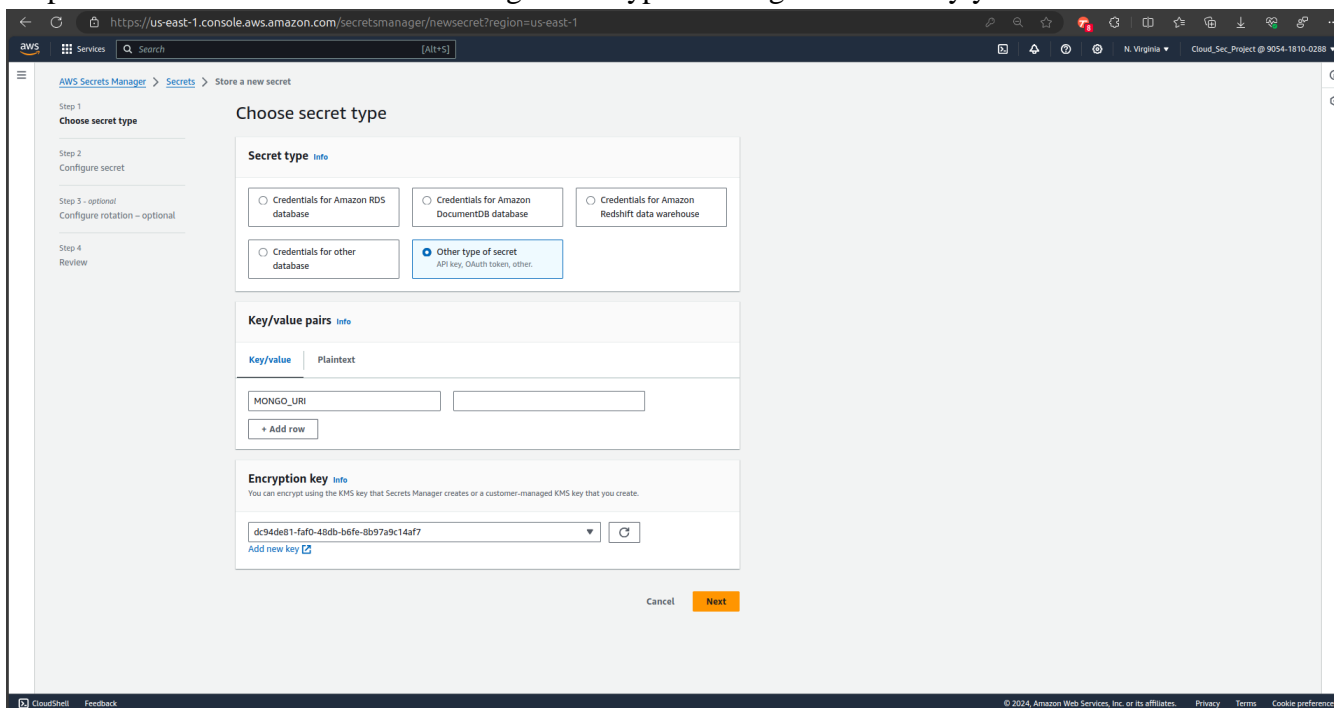
6. Storing Application Secrets in Secrets Manager

Step 1: Navigate to AWS Secrets Manager.

Step 2: Click "Store a new secret."

Step 3: Select Other type of secret and provide key-pair.

Step 4: Enter the secret value and configure encryption using the KMS key you created.



Step 1
Choose secret type

Step 2
Configure secret

Step 3 - optional
Configure rotation - optional

Step 4
Review

Configure secret

Secret name and description [info](#)

Secret name
A descriptive name that helps you find your secret later.
Loan-app-secrets
Secret name must only contain alphanumeric characters and the characters /, *, @.

Description - optional
e.g. Access to MySQL prod database for my AppIeta
Maximum 250 characters.

Tags - optional
No tags associated with the secret.
[Add](#)

Resource permissions - optional [info](#) [Edit permissions](#)
Add or edit a resource policy to access secrets across AWS accounts.

► **Replicate secret - optional**
Create read-only replicas of your secret in other regions. Replica secrets incur a charge.

[Cancel](#) [Previous](#) [Next](#)

7. ECS Setup for Application

Step 1: Navigate to the ECS dashboard and select "Clusters."

Tell us what you think

Amazon Elastic Container Service

Clusters

Namespaces

Task definitions

Account settings

Install AWS Copilot

Amazon ECR

Repositories

AWS Batch

Documentation

Discover products

Subscriptions

Amazon Elastic Container Service > Create cluster

Create cluster

An Amazon ECS cluster groups together tasks and services, and allows for shared capacity and common configurations. All of your tasks, services and capacity must belong to a cluster.

Cluster configuration

Cluster name
LoanAppCluster
Cluster name must be 1 to 255 characters. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).

Default namespace - optional
Select the namespace to specify a group of services that make up your application. You can overwrite this value at the service level.
LoanAppCluster

▼ **Infrastructure** [info](#) [Serverless](#)
Your cluster is automatically configured for AWS Fargate (serverless) with two capacity providers. Add Amazon EC2 instances, or external instances using ECS Anywhere.

☒ **AWS Fargate (serverless)**
Pay as you go. Use if you have tiny, batch or burst workloads or for zero maintenance overhead. The cluster has Fargate and Fargate Spot capacity providers by default.

☐ **Amazon EC2 instances**
Manual configurations. Use for large workloads with consistent resource demands.

☐ **External instances using ECS Anywhere**
Manual configurations. Use to add data centre compute.

► **Monitoring - optional** [info](#)
Container Insights is turned off by default. To change the default behaviour, use the CloudWatch Container Insights account setting. When you use Container Insights, there is a cost associated with it.

► **Tags - optional** [info](#)
Tags help you to identify and organise your clusters.

Step 2: Click "Create Cluster," choose Fargate, and configure the required settings.

Step 3: Create a task definition where you specify the Docker image to use (from ECR), CPU and memory configurations, and the IAM role for the task.

← → ↻ https://us-east-1.console.aws.amazon.com/ecs/v2/task-definitions/cloudproject-tf/1/create-revision?region=us-east-1

AWS Services Search [Alt+S]

Tell us what you think X

Amazon Elastic Container Service

Clusters
Namespaces
Task definitions
Account settings

Install AWS Copilot [?](#)

Amazon ECR [?](#)
Repositories

AWS Batch [?](#)

Documentation [?](#)
Discover products [?](#)
Subscriptions [?](#)

Amazon Elastic Container Service > Task definitions > cloudproject-tf > Revision 1 > Create revision

Create new task definition revision [Info](#)

Task definition configuration

Task definition family [Info](#)
Specify a unique task definition family name.
cloudproject-tf
Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

Revision
Source revision
1

Infrastructure requirements
Specify the infrastructure requirements for the task definition.

Launch type [Info](#)
Selection of the launch type will change task definition parameters.
☒ **AWS Fargate**
Serverless compute for containers.

☐ **Amazon EC2 instances**
Self-managed infrastructure using Amazon EC2 instances.

OS, Architecture, Network mode
Network mode is used for tasks and is dependent on the compute type selected.
Operating system/Architecture [Info](#)
Choose the operating system or arch... ▾

Network mode [Info](#)
awsipc ▾

Task size [Info](#)
Specify the amount of CPU and memory to reserve for your task.
CPU
.5 vCPU ▾
Memory
1 GB ▾

Task roles - conditional

Task role [Info](#)
A task IAM role allows containers in the task to make API requests to AWS services. You can create a task IAM role from the IAM console [?](#)

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

← → ↻ https://us-east-1.console.aws.amazon.com/ecs/v2/task-definitions/cloudproject-tf/1/create-revision?region=us-east-1

AWS Services Search [Alt+S]

Tell us what you think X

Amazon Elastic Container Service

Clusters
Namespaces
Task definitions
Account settings

Install AWS Copilot [?](#)

Amazon ECR [?](#)
Repositories

AWS Batch [?](#)

Documentation [?](#)
Discover products [?](#)
Subscriptions [?](#)

Amazon Elastic Container Service > Task definitions > cloudproject-tf > Revision 1 > Create revision

Create new task definition revision [Info](#)

Container - 1 [Info](#) Essential container Remove

Container details
Specify a name, container image and whether the container should be marked as essential. Each task definition must have at least one essential container.
Name app
Image URI 905418100288.dkr.ecr.us-east-1.amazonaws.com/loan-app-cloudprojectlate
Essential container Yes ▾

Private registry [Info](#)
Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.
☒ **Private registry authentication**

Port mappings [Info](#)
Add port mappings to allow the container to access ports on the host to send or receive traffic. For port name, a default will be assigned if left blank.

Container port	Protocol	Port name	App protocol	
5000	TCP ▾	container-port-protocol	App protocol ▾	Remove

Add port mapping

Read-only root file system [Info](#)
When this parameter is turned on, the container is given read-only access to its root file system.
☐ **Read only**

Resource allocation limits - conditional [Info](#)
Container-level CPU, GPU and memory limits are different from task-level values. They define how many resources are allocated for the container. If the container attempts to exceed the memory specified by the hard limit, the container is terminated.

CPU	GPU	Memory hard limit	Memory soft limit
0.5 <small>in vCPU</small>	1	1 <small>in GB</small>	1 <small>in GB</small>

Environment variables - optional

Environment variables [Info](#)
Add individually
Add a key-value pair to specify an environment variable.

Key	Value type	Value	
JWT_SECRET	ValueFrom ▾	arn:aws:secretsmanager:	Remove
MONGO_URI	ValueFrom ▾	arn:aws:secretsmanager:	Remove

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step 4: Launch a service within the cluster that specifies the number of desired tasks and networking settings.

The image displays two screenshots of the AWS Management Console, specifically the 'Create Service' wizard for Amazon ECS in the us-east-1 region.

Top Screenshot: Compute configuration (advanced)

- Compute options:**
 - Capacity provider strategy:** (disabled)
 - Launch type:** Selected. Launch tasks directly without the use of a capacity provider strategy.
- Launch type:** FARGATE
- Platform version:** LATEST
- Deployment configuration:**
 - Application type:** Service (selected). Launch a group of tasks handling a long-running computing work that can be stopped and restarted. For example, a web application.
 - Task definition:** cloudproject-tf, Revision 1 (LATEST)
 - Service name:** loan-app-service
 - Service type:** Replica

Bottom Screenshot: Networking

- VPC:** vpc-06f1b768a573b4f2f (cloudproject_vpc)
- Subnets:**
 - private_subnet-5 (us-east-1c 10.0.5.0/24)
 - private_subnet-1 (us-east-1a 10.0.3.0/24)
 - public_subnet-5 (us-east-1c 10.0.2.0/24)
 - private_subnet-2 (us-east-1b 10.0.4.0/24)
 - public_subnet-2 (us-east-1b 10.0.1.0/24)
 - public_subnet-1 (us-east-1a 10.0.0.0/24)
- Security group:** sg-07cdf399532dae505 (default)
- Public IP:** Turned on

8. Creating Application Load Balancer and Attaching It to ECS Service

Step 1: Go to the EC2 service, select "Load Balancers" and then "Create Load Balancer."

Step 2: Choose "Application Load Balancer," set VPC and subnets.

← <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#SelectCreateELBWizard>

Services Search [Alt+S]

EC2 > Load balancers > Compare and select load balancer type

Compare and select load balancer type

A complete feature-by-feature comparison along with detailed highlights is also available. [Learn more](#)

Load balancer types

Application Load Balancer

Choose an Application Load Balancer when you need a flexible feature set for your applications with HTTP and HTTPS traffic. Operating at the request level, Application Load Balancers provide advanced routing and visibility features targeted at application architectures, including microservices and containers.

Create

Network Load Balancer

Choose a Network Load Balancer when you need ultra-high performance, TLS offloading at scale, centralized certificate deployment, support for UDP, and static IP addresses for your applications. Operating at the connection level, Network Load Balancers are capable of handling millions of requests per second securely while maintaining ultra-low latencies.

Create

Gateway Load Balancer

Choose a Gateway Load Balancer when you need to deploy and manage a fleet of third-party virtual appliances that support GENEVE. These appliances enable you to improve security, compliance, and policy controls.

Create

▶ Classic Load Balancer - previous generation

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

← <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateALBWizard>

Services Search [Alt+S]

Basic configuration

Load balancer name

Name must be unique within your AWS account and can't be changed after the load balancer is created.

loan-alb

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Scheme

Scheme can't be changed after the load balancer is created.

☒ Internet-facing

An internet-facing load balancer routes requests from clients over the Internet to targets. Requires a public subnet. [Learn more](#)

☐ Internal

An internal load balancer routes requests from clients to targets using private IP addresses.

IP address type

Select the type of IP addresses that your subnets use.

☒ IPv4

Includes only IPv4 addresses.

☐ Dualstack

Includes IPv4 and IPv6 addresses.

Network mapping

The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.

VPC

Select the virtual private cloud (VPC) for your targets or you can [create a new VPC](#). Only VPCs with an Internet gateway are enabled for selection. The selected VPC can't be changed after the load balancer is created. To confirm the VPC for your targets, view your [target groups](#).

cloudproject_vpc

vpc-0671b76da573a42f

IPv4 VPC CIDR: 10.0.0.0/16

Mappings

Select at least two Availability Zones and one subnet per zone. The load balancer routes traffic to targets in these Availability Zones only. Availability Zones that are not supported by the load balancer or the VPC are not available for selection.

☒ us-east-1a (use1-az6)

Subnet

subnet-009d5a77a6943bda public_subnet-1

IPv4 address

Assigned by AWS

☐ us-east-1b (use1-az1)

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step 3: Create Target group.

Step 4: Select IP instances for targets to be registered.

← <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:protocol=HTTP:vpc=vpc-06f1b768a573b4f2f>

Services Search [Alt+S]

Specify group details

Step 2: Register targets

Your load balancer routes requests to the targets in a target group and performs health checks on the targets.

Basic configuration

Settings in this section can't be changed after the target group is created.

Choose a target type

☐ Instances

- Supports load balancing to instances within a specific VPC.
- Facilitates the use of [Amazon EC2 Auto Scaling](#) to manage and scale your EC2 capacity.

☒ IP addresses

- Supports load balancing to VPC and on-premises resources.
- Facilitates routing to multiple IP addresses and network interfaces on the same instance.
- Offers flexibility with microservice based architectures, simplifying inter-application communication.
- Supports IPv6 targets, enabling end-to-end IPv6 communication, and IPv4-to-IPv6 NAT.

☐ Lambda function

- Facilitates routing to a single Lambda function.
- Accessible to Application Load Balancers only.

☐ Application Load Balancer

- Offers the flexibility for a Network Load Balancer to accept and route TCP requests within a specific VPC.
- Facilitates using static IP addresses and PrivateLink with an Application Load Balancer.

Target group name

loan-app-tg

A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.

Protocol: Port

Choose a protocol for your target group that corresponds to the Load Balancer type that will route traffic to it. Some protocols now include anomaly detection for the targets and you can set mitigation options once your target group is created. This choice cannot be changed after creation.

HTTP 80

1-65535

IP address type

Only targets with the indicated IP address type can be registered to this target group.

☒ IPv4

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

Step 5: Click on “Next” and leave everything default and click on create target group.

← <https://us-east-1.console.aws.amazon.com/ec2/home?region=us-east-1#CreateTargetGroup:protocol=HTTP:vpc=vpc-06f1b768a573b4f2f>

Services Search [Alt+S]

vpc-06f1b768a573b4f2f
IPv4 VPC CIDR: 10.0.0.0/16

Step 2: Specify IPs and define ports

You can manually enter IP addresses from the selected network.

Enter an IPv4 address from a VPC subnet.

10.0.0.

Remove

Add IPv4 address

You can add up to 4 more IP addresses.

Ports

Ports for routing to this target.

80

1-65535 (separate multiple ports with comma)

Include as pending below

Review targets

Step 3: Review IP targets to include in your group

Confirm the IP targets to include in your target group. Add more IP targets by repeating steps 1 and 2 on this page. You can also register additional targets after your target group is created.

Targets (0)

Filter targets

Show only pending

Remove all pending

Remove IPv4 address	Health status	IP address	Port	Zone
No IP addresses included yet Specify IP addresses above and add to list.				

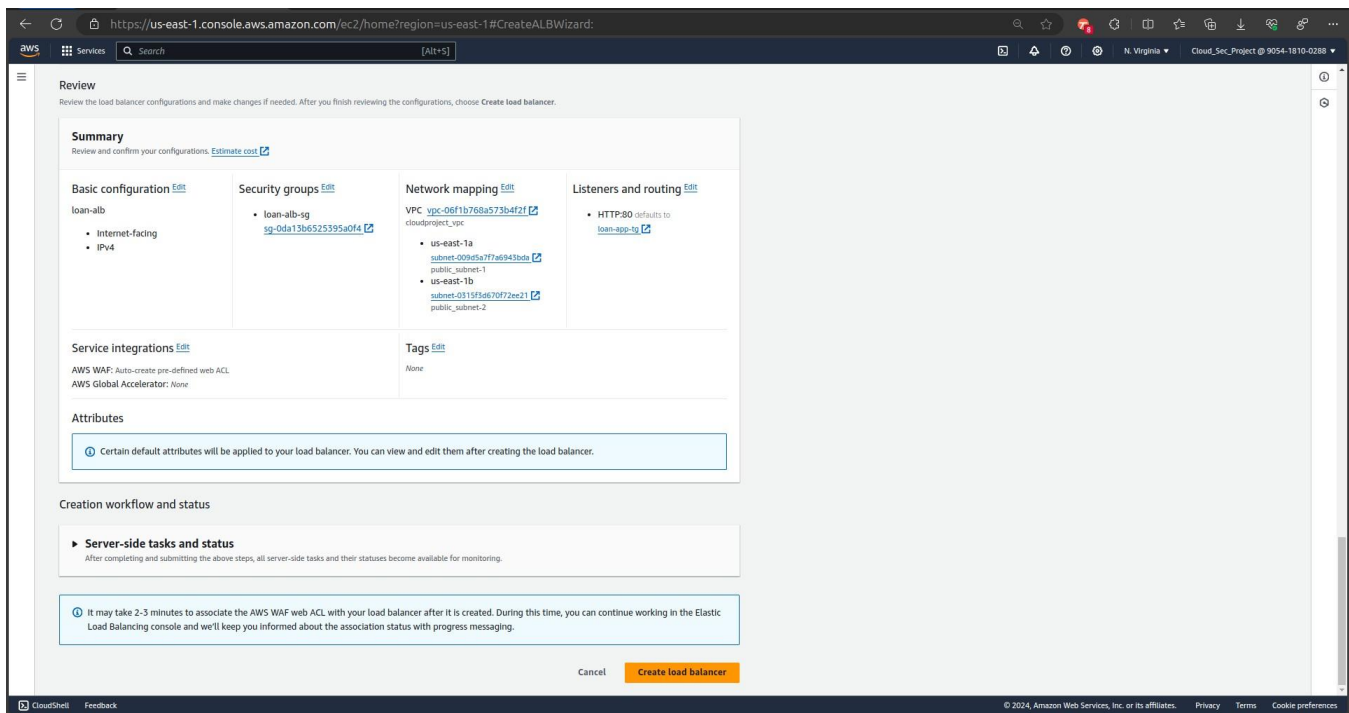
0 pending

Cancel Previous Create target group

CloudShell Feedback

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

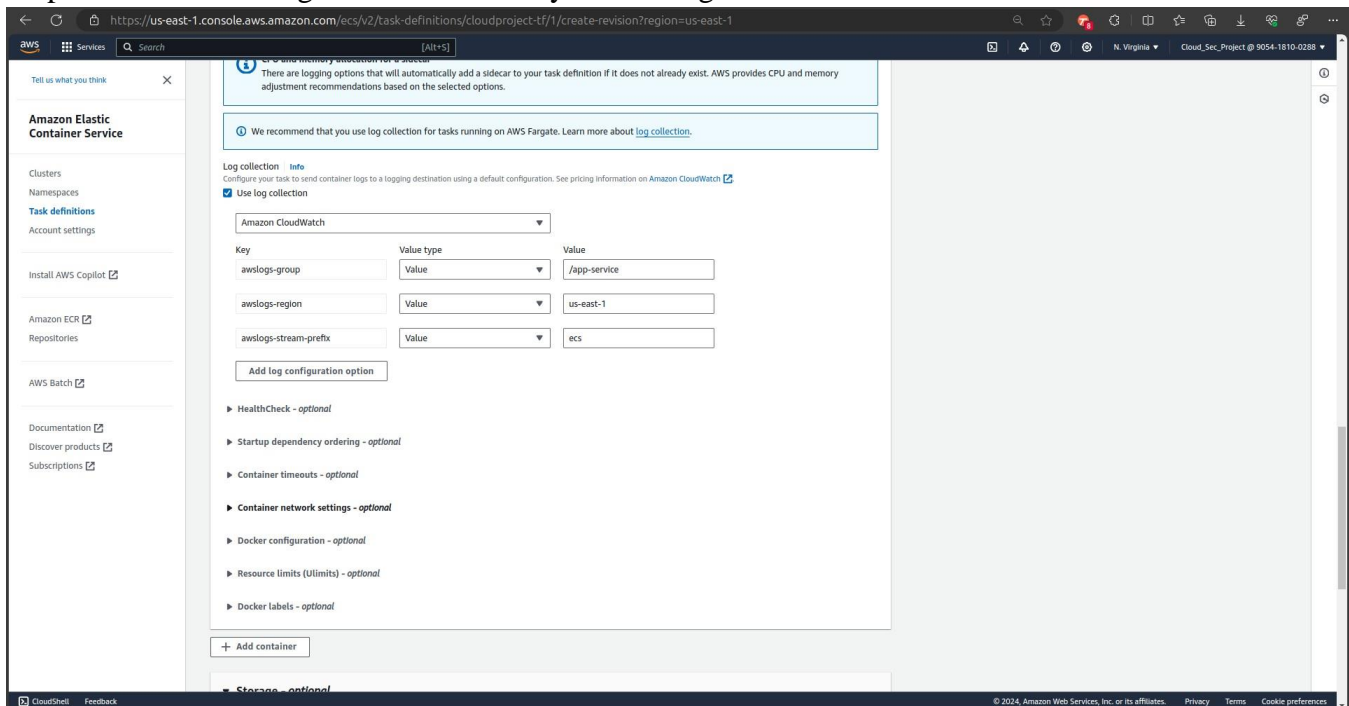
Link it to your ECS service by designating the target group in the ECS service definition and click on Create Load Balancer.



9. Creating CloudWatch Logs for ECS Containers

Step 1: Within the ECS task definition, configure the log configuration to use "awslogs" and set the appropriate log group and region in CloudWatch.

Step 2: CloudWatch logs will automatically receive logs from the ECS containers as defined.

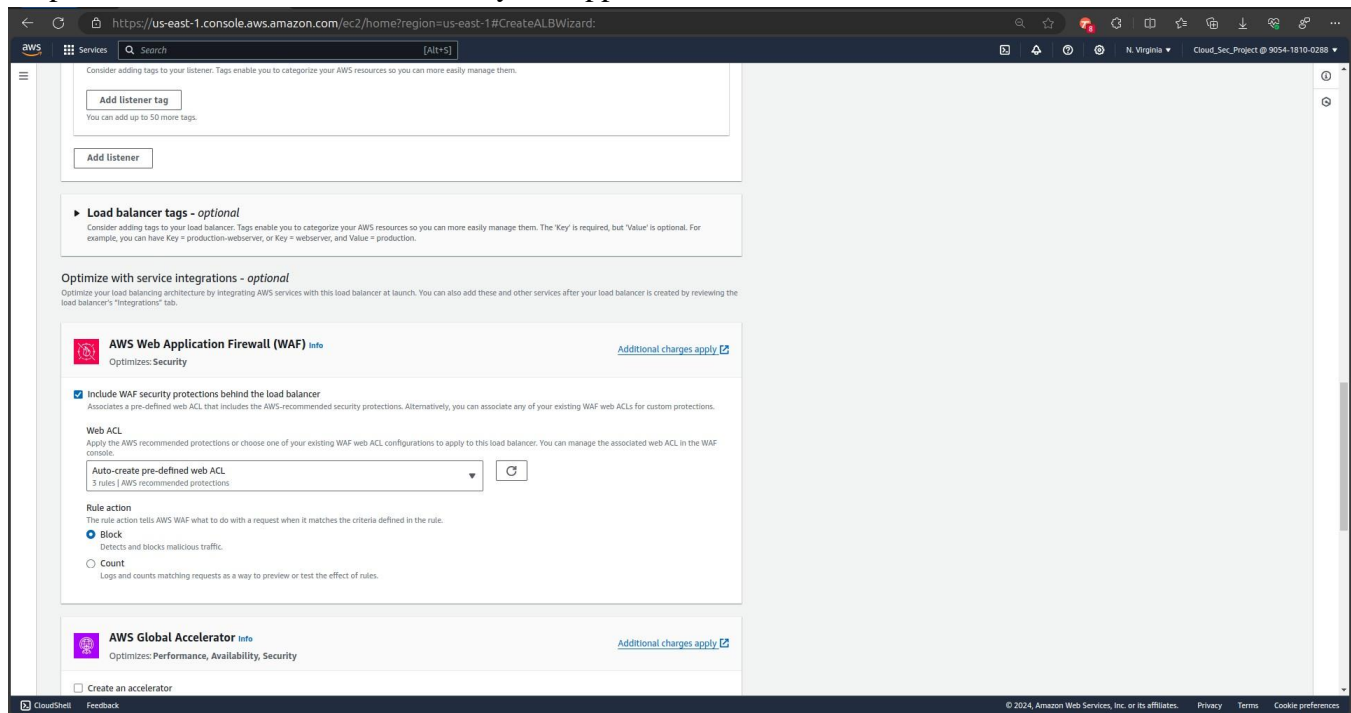


10. Attaching WAF to the Load Balancer

Step 1: Navigate to AWS WAF & Shield in the AWS Console.

Step 2: Create a Web ACL and define rules to manage incoming traffic.

Step 3: Associate this Web ACL with your Application Load Balancer.



Learnings

Secure Storage Practices with AWS Secrets Manager

Through the implementation of AWS Secrets Manager, the project spotlighted the critical importance of securely storing and managing application secrets. Prior to using Secrets Manager, secrets might have been embedded in code or configuration files, which posed a significant risk. However, with Secrets Manager, we learned how to centralize secret storage and rotate credentials automatically, which significantly mitigates the risk of data breaches. This practice is not just about hiding secrets but also about managing their lifecycle from creation to retirement, ensuring that they are always inaccessible to unauthorized entities.

Encryption Best Practices with AWS KMS

Employing AWS Key Management Service (KMS) introduced us to the best practices in data encryption. KMS allowed us to create and control encryption keys used to encrypt data, offering a robust management system that integrates with other AWS services to encrypt data across the platform. We learned the importance of using these tools to maintain encryption standards for data at rest and in transit, ensuring sensitive information is unreadable to unauthorized users and protected from potential breaches.

Advanced Monitoring with AWS CloudWatch

AWS CloudWatch has been instrumental in elevating our monitoring capabilities. It serves as a comprehensive observatory platform, granting us the ability to watch over the application and infrastructure metrics, set alarms, and react to changes in our environment in real time. This advanced

monitoring helped us move from a reactive to a proactive stance on performance issues, allowing us to anticipate and address bottlenecks before they impact the user experience. CloudWatch logs also provide a granular view of system operations, which is invaluable for troubleshooting and understanding system behavior under various load conditions.

Scalability via AWS Load Balancers

Integrating AWS Load Balancers has taught us about the dynamic nature of web traffic and the need for elastic scalability. Load balancers distribute incoming application traffic across multiple targets, such as EC2 instances, ensuring no single instance is overwhelmed. This not only aids in handling traffic surges gracefully but also contributes to fault tolerance and uninterrupted service availability. We learned how to scale our application horizontally and leverage health checks to route traffic away from faulty instances, ensuring continuous availability.

Protection Against Web Attacks with AWS WAF

The deployment of AWS Web Application Firewall (WAF) brought to light the intricacies of protecting web applications from external threats. AWS WAF acts as a shield, filtering out malicious traffic before it reaches our application. Through the use of WAF, we learned how to set up custom web security rules that address the OWASP Top 10 security risks and more, which can be tailored to the specific needs of our application. WAF's capabilities taught us about the patterns of web attacks and how to respond to them effectively, significantly strengthening our security posture.

Networking Security with AWS Security Groups

The configuration of AWS Security Groups played a crucial role in teaching us the finer points of network security within a cloud environment. Security Groups act as virtual firewalls that regulate inbound and outbound traffic to services like EC2 instances. We learned how to define and meticulously enforce firewall rules that allow only the necessary traffic for the application to function, while blocking all other traffic, which greatly enhances the overall security of our infrastructure.

Challenges

Navigating the Complexity of AWS Services

Challenge:

Our journey into the AWS ecosystem was like venturing into a labyrinth of powerful yet complex services. The initial challenge was to understand and effectively integrate a constellation of services, including ECS, ECR, and EC2. This task demanded an intricate understanding of how these services interact and complement each other to form a cohesive, seamless system.

Response:

To navigate this complexity, we invested in the most valuable resource—knowledge. Our team engaged in AWS training and certification programs, which acted as a compass to guide us through the AWS maze. We complemented formal learning with the rich wisdom of the AWS community forums and pored over extensive documentation. This approach sharpened our skills and empowered us to architect a system where all components function in symphony, providing a robust and reliable platform for our users.

Scalability and Performance Optimization

Challenge:

As our platform stretched its wings to accommodate a growing flock of users, we grappled with the challenge of scalability. The system needed to expand and contract with the ebb and flow of demand, especially during the high tide of peak operational hours. Balancing scalability with performance was a high-wire act, where the safety net of user experience could not be compromised.

Response:

We harnessed the power of AWS ECS, which offered sophisticated load management capabilities. It was like employing a skilled team of traffic controllers, directing the flow of data to ensure smooth operations. The AWS Load Balancer served as a roundabout, managing the influx of requests by distributing them efficiently across our resources. Meanwhile, AWS CloudWatch stood as our vigilant lookout, continuously monitoring the horizon for performance metrics. Its real-time insights enabled us to adjust our resources dynamically, akin to a conductor modulating the tempo of an orchestra to maintain a harmonious performance.

Conclusion:

"Loan Management with AWS" revolutionize loan management in the financial sector. Through a meticulously designed architecture utilizing Flask, MongoDB, and a suite of AWS services like ECS, ECR, and WAF, the project addresses common industry challenges while enhancing scalability, security, and user experience. By embracing best practices in data protection, scalability optimization, and proactive monitoring, this demonstrates a profound understanding of cloud technology and its potential to reshape financial services. With its commitment to innovation and excellence to empower both lenders and borrowers in the digital era.