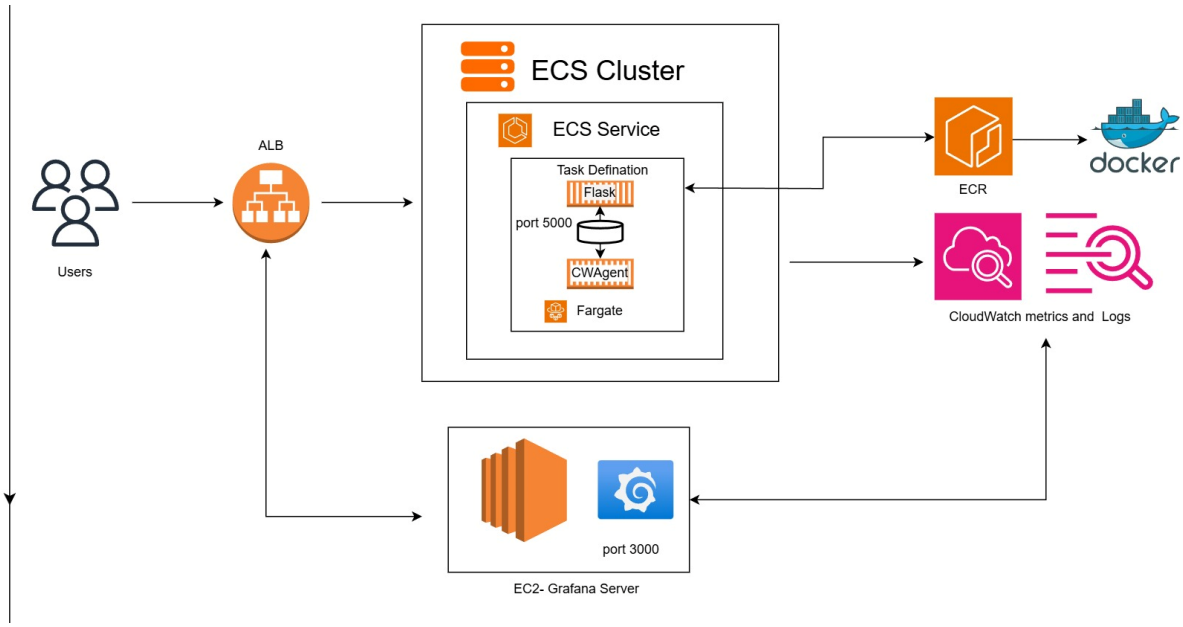
Run an application on ECS, push the application logs to CloudWatch, and use the CloudWatch Agent as a sidecar container. Also, install Grafana on ECS and display all the application metrics in Grafana.

Description:

- **ECS Cluster Setup**: You have an ECS cluster that contains one service.
- **Task Configuration**: This service is running a single task that contains two containers:
  - **Flask Application**: This container is running a Flask app, serving on port 5000.
  - **CloudWatch Agent (Sidecar)**: This container is configured as a sidecar, responsible for collecting logs and metrics from the Flask app, sending them to AWS CloudWatch.
- **Fargate**: Both containers are running in AWS Fargate, a serverless compute engine that eliminates the need to manage the underlying infrastructure.
- **Application Load Balancer (ALB)**: The ECS service (and containers) is connected to an ALB, which handles traffic routing to your Flask app. The ALB can route HTTP requests to the Flask container running on port 5000.
- **Auto Scaling Group (ASG)**: Your ECS service is configured with Auto Scaling to adjust the number of running tasks based on load, helping to handle variable traffic.
- **Grafana Instance**: You have a separate EC2 instance running Grafana, which is attached to the same ALB. Grafana is used to visualize and analyze logs and metrics from CloudWatch.

Architecture Diagram:



For this task I am using simple flask application as main container and cloud watch agent as sidecar container.

Flask and Cloudwatch agent code with Dockerfile.

```
/flask-multipage-app
│
├── app/                        # Flask application directory
│   ├── app.py                  # Flask app entry point
│   ├── requirements.txt        # Python dependencies
│   ├── templates/              # HTML templates
│   │   ├── index.html
│   │   ├── about.html
│   │   ├── contact.html
│   │   └── 404.html
│   └── static/                 # Static files (e.g., CSS)
│       └── style.css
│
├── cloudwatch-agent/           # CloudWatch Agent directory
│   ├── Dockerfile              # Dockerfile for CloudWatch Agent
│   └── cloudwatch-agent-config.json # CloudWatch Agent config file
└── logs/                       # Directory for application logs (shared volume)
```

File structure.

Github link for the code: https://github.com/Koushikshivu/AWS-ECS-Sidecar.

Build and push both images to ECR.

○	cw-agent	183631336766.dkr.ecr.ap-south-1.amazonaws.com/cw-agent	January 23, 2025, 12:55:35 (UTC+05.5)	Mutable	AES-256
○	simple-flask-app	183631336766.dkr.ecr.ap-south-1.amazonaws.com/simple-flask-app	January 25, 2025, 15:53:01 (UTC+05.5)	Mutable	AES-256

Create ECS Task Definition.

Using fargate.

Launch type | Info
Selection of the launch type will change task definition parameters.

- [x] 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: Linux/X86_64

Network mode | Info: awsvpc

Task size | Info
Specify the amount of CPU and memory to reserve for your task.

CPU: .25 vCPU

Memory: .5 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.

ecsTaskExecutionRole

Task execution role | Info
A task execution IAM role is used by the container agent to make AWS API requests on your behalf. If you don't already have a task execution IAM role created, we can create one for you.

ecsTaskExecutionRole

Container1 add flask app image id and add container port.

▼ **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: flask-container
Up to 255 letters (uppercase and lowercase), numbers, hyphens, and underscores are allowed.

Image URI: 183631336766.dkr.ecr.ap-south-1.amazonaws.com/simple-flask-app:latest
Up to 255 letters (uppercase and lowercase), numbers, hyphens, underscores, colons, periods, forward slashes, and number signs are allowed.

Essential container: Yes

Private registry | Info
Store credentials in Secrets Manager, and then use the credentials to reference images in private registries.

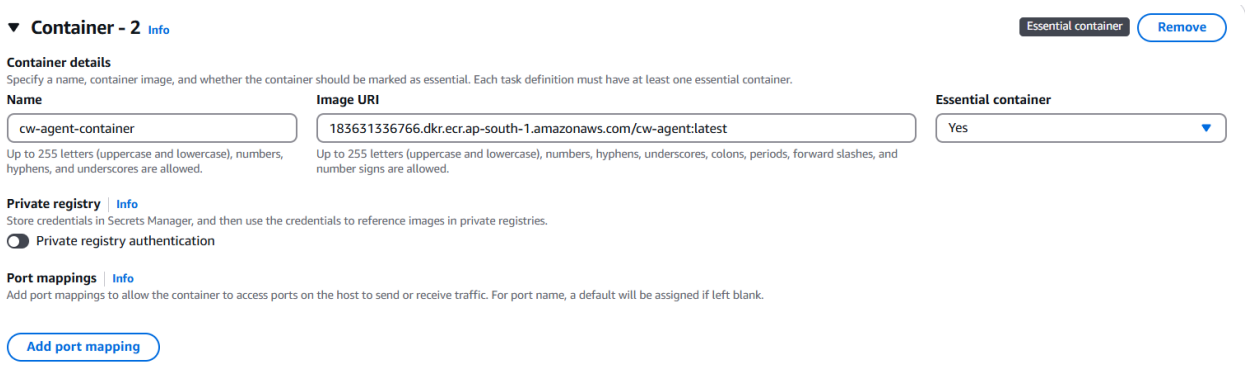
Private registry authentication

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.

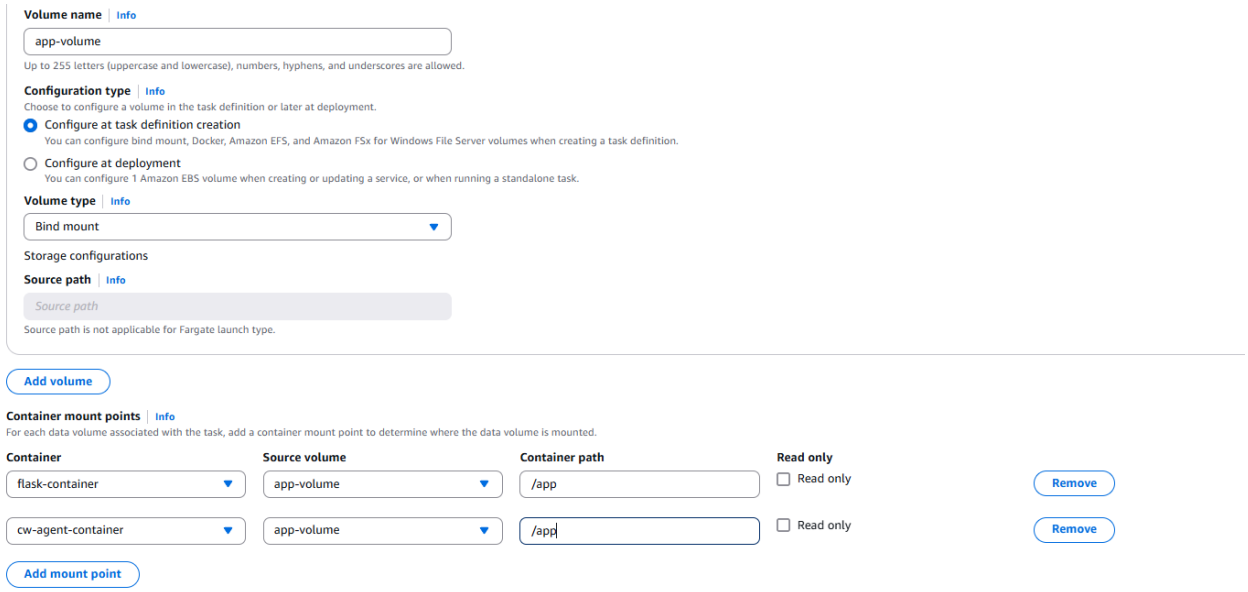
Container port	Protocol	Port name	App protocol	
5000	TCP	container-port-protocol	HTTP	Remove

Add port mapping

Container2 add Cloudwatch agent image id.

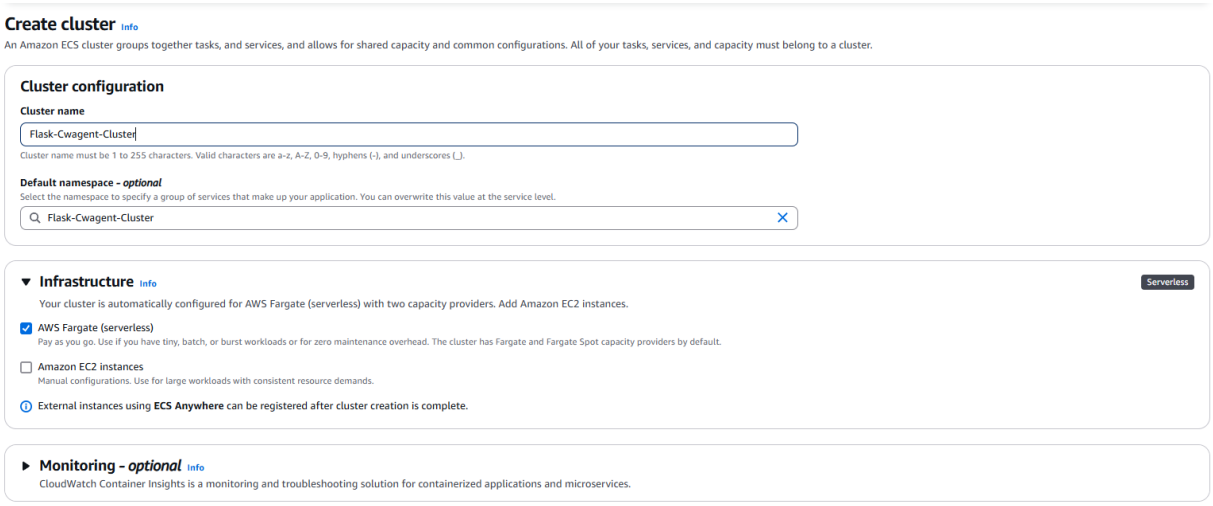


Create a shared Volume and add the container path.

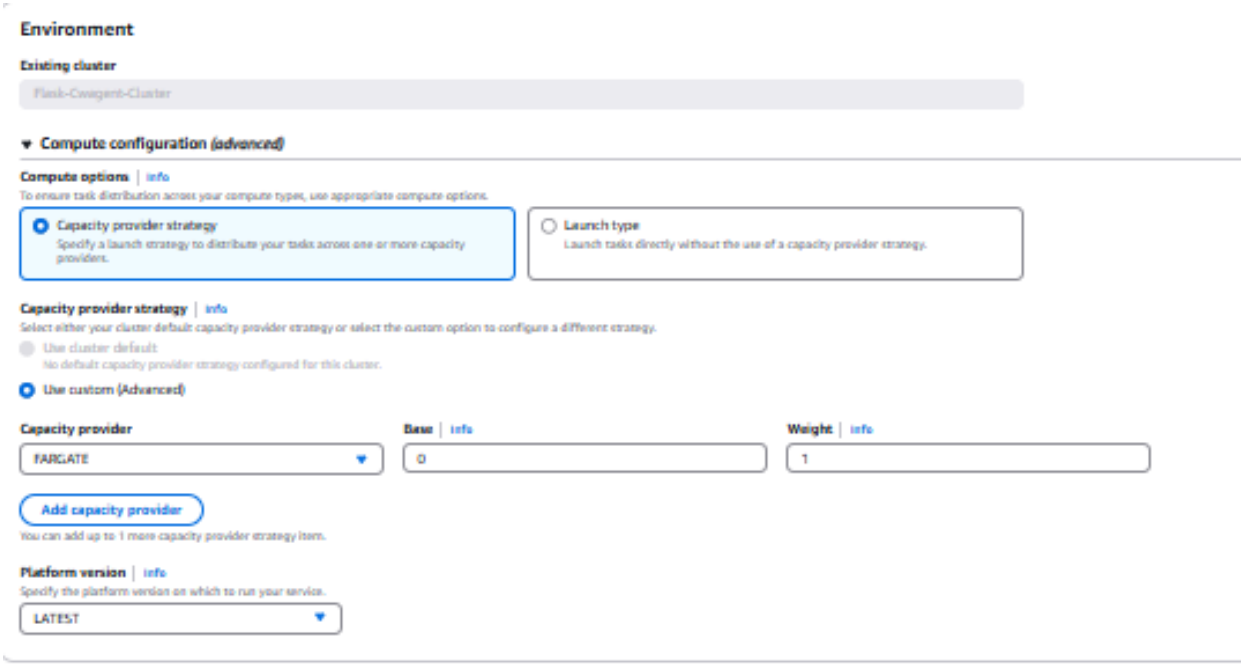


Note: flask application logs will collected in /app folder stored in *.log file where CWAgent will push the logs to Cloudwatch log groups for that both container should have same volume mounted to /app folder.

Creating ECS Cluster.



Service Creation steps:



Deployment configuration

Application type | Info

Specify what type of application you want to run.

- (●) **Service** — Launch a group of tasks handling a long-running computing work that can be stopped and restarted. For example, a web application.
- ( ) **Task** — Launch a standalone task that runs and terminates. For example, a batch job.

Task definition

Select an existing task definition. To create a new task definition, go to Task definitions.

[ ] Specify the revision manually

Manually input the revision instead of choosing from the 100 most recent revisions for the selected task definition family.

Family: flask-cwagent-sidecar-task

Revision: 1 (LATEST)

Service name

Assign a service name that is unique for this cluster.

flask-cwagent-service

Up to 255 letters (uppercase and lowercase), numbers, underscores, and hyphens are allowed. Service names must be unique within a cluster.

Service type | Info

Specify the service type that the service scheduler will follow.

- (●) **Replica** — Place and maintain a desired number of tasks across your cluster.
- ( ) **Daemon** — Place and maintain one copy of your task on each container instance.

Desired tasks

Specify the number of tasks to launch.

1

Availability Zone rebalancing | Info

[x] Turn on Availability Zone rebalancing

Amazon ECS automatically detects Availability Zone imbalances in task distributions across an ECS service, and evenly redistributes ECS service tasks across Availability Zones.

Create ALB and ASG for server to expose the application.

Service auto scaling - *optional*

Automatically adjust your service's desired count up and down within a specified range in response to CloudWatch alarms. You can modify your service auto scaling configuration at any time to meet the needs of your application.

[x] Use service auto scaling

Configure service auto scaling to adjust your service's desired count

Minimum number of tasks

The lower boundary to which service auto scaling can adjust the desired count of the service.

1

Maximum number of tasks

The upper boundary to which service auto scaling can adjust the desired count of the service.

5

Scaling policy type | Info

Create either a target tracking or step scaling policy.

- (●) **Target tracking** — Increase or decrease the number of tasks that your service runs based on a target value for a specific metric.
- ( ) **Step scaling** — Increase or decrease the number of tasks that your service runs based on a set of scaling adjustments, known as step adjustments, that vary based on the size of the alarm breach.

Policy name

flask-cwagent-ASG-policy

ECS service metric

ECSServiceAverageCPUUtilization

Target value

70

Scale-out cooldown period

300

Scale-in cooldown period

300

[ ] Turn off scale-in

Load balancing - *optional*

Configure load balancing using Amazon Elastic Load Balancing to distribute traffic evenly across the healthy tasks in your service.

☑ Use load balancing

Load balancer type | Info

Specify the load balancer type to distribute incoming traffic across the tasks running in your service.

- (●) **Application Load Balancer** — An Application Load Balancer makes routing decisions at the application layer (HTTP/HTTPS), supports path-based routing, and can route requests to one or more ports.
- ( ) **Network Load Balancer** — A Network Load Balancer makes routing decisions at the transport layer (TCP/UDP).

Container

The container and port to load balance the incoming traffic to

flask-container 5000:5000

Host port:Container port

Application Load Balancer

Specify whether to create a new load balancer or choose an existing one.

- (●) Create a new load balancer
- ( ) Use an existing load balancer

Load balancer name

Assign a unique name for the load balancer.

flask-cwagent-ALB

Health check grace period | Info

0

seconds

Listener | Info

Specify the port and protocol that the load balancer will listen for connection requests on.

- (●) Create new listener
- ( ) Use an existing listener — You need to select an existing load balancer.

Port

80

Protocol

HTTP

Target group | Info

Specify whether to create a new target group or choose an existing one that the load balancer will use to route requests to the tasks in your service.

- (●) Create new target group
- ( ) Use an existing target group

Target group name

ecs-Flask--flask-cwagent-service

Protocol

HTTP

Deregistration delay

The amount of time to wait before the state of a deregistering target changes from draining to unused.

300

seconds

Health check protocol

HTTP

Health check path | Info

/

Load balancers (1/1)

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

Actions ▾ | Create load balancer ▾

Filter load balancers

1

Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
flask-cwagent-alb	flask-cwagent-alb-780855200.ap-south-1.elb.amazonaws.com	Active	vpc-0195dc72432e611af	3 Availability Zones	application	January 26, 2025, 18:

Load balancer: flask-cwagent-alb

flask-cwagent-alb

Last fetched seconds ago | Export

Listeners (2)

- HTTP:3000 — 1 rule
- HTTP:80 — 1 rule

Rules (2)

- Priority default — Forward to target group
  Conditions (If)
  If no other rule applies
- Priority default — Forward to target group
  Conditions (If)
  If no other rule applies

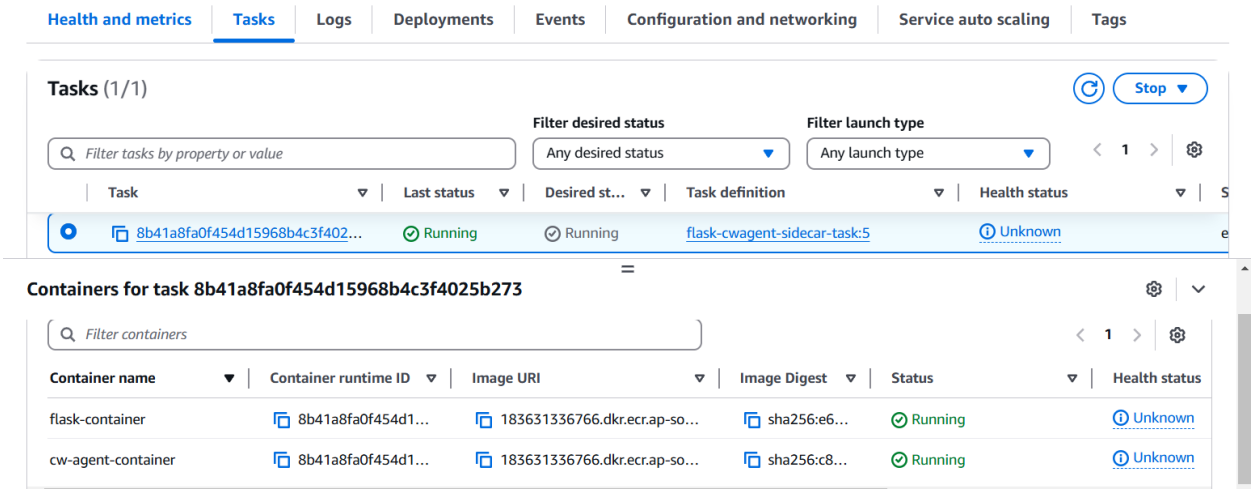
Target groups (2) Info

- Instance — Grafana — 1 target
  1 | 0 | 0 | 0 | 0
- IP — ecs-flask--flask-cwagent-service — 1 target
  1 | 0 | 0 | 0 | 0

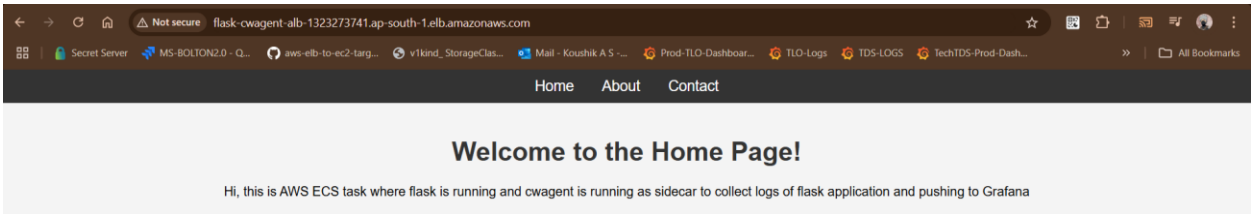
Targets (2)

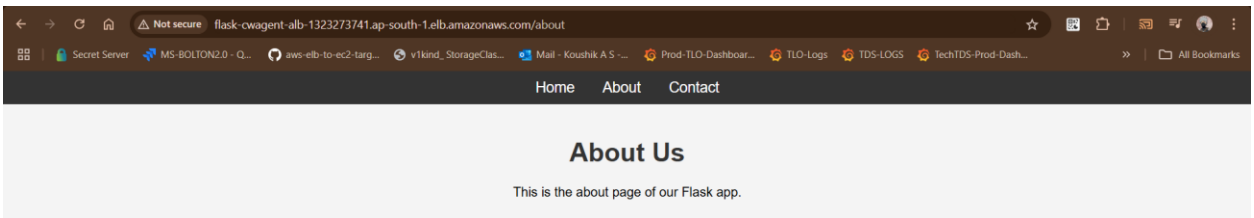
- i-0b69062c7b1372c18 — Port 3000 — Healthy
- 172.31.47.44 — Port 5000 — Healthy

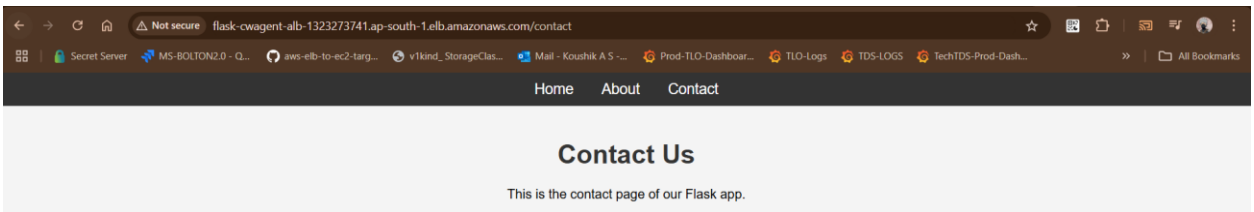
Task created with 2 Container flask and cwagent.



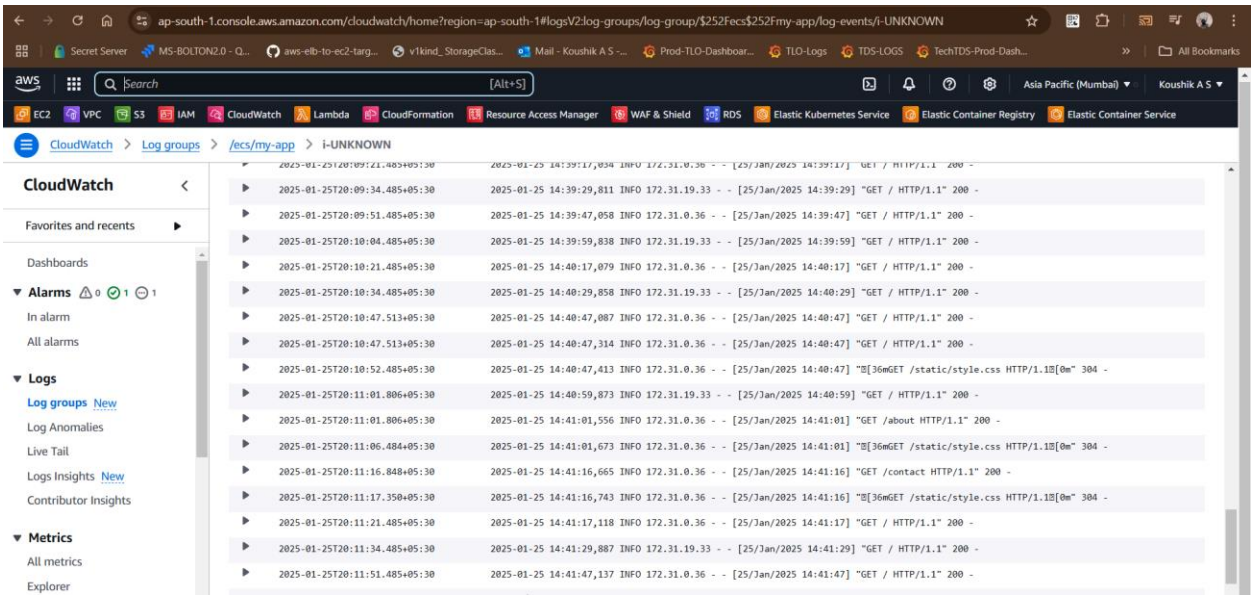
Check by hitting ALB DNS.



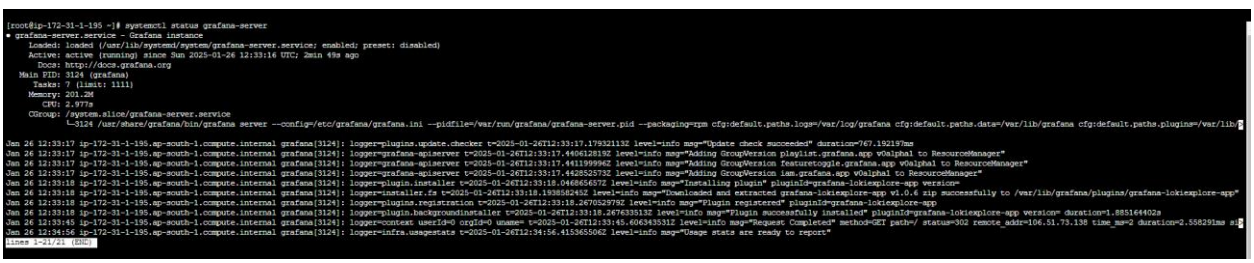




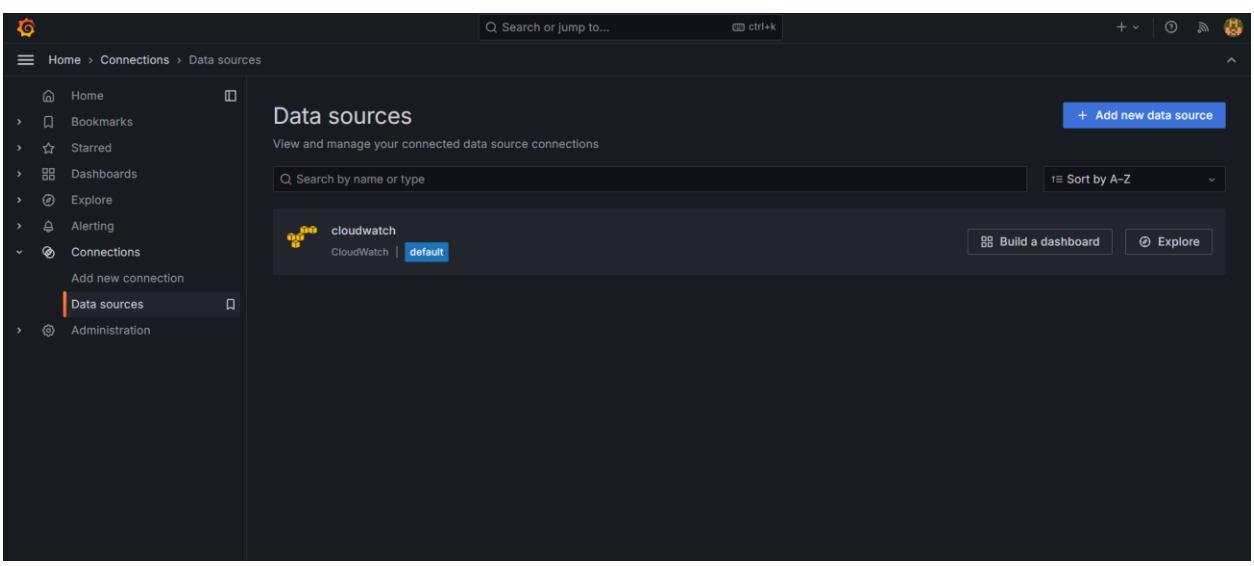
Check Cloud watch logs.



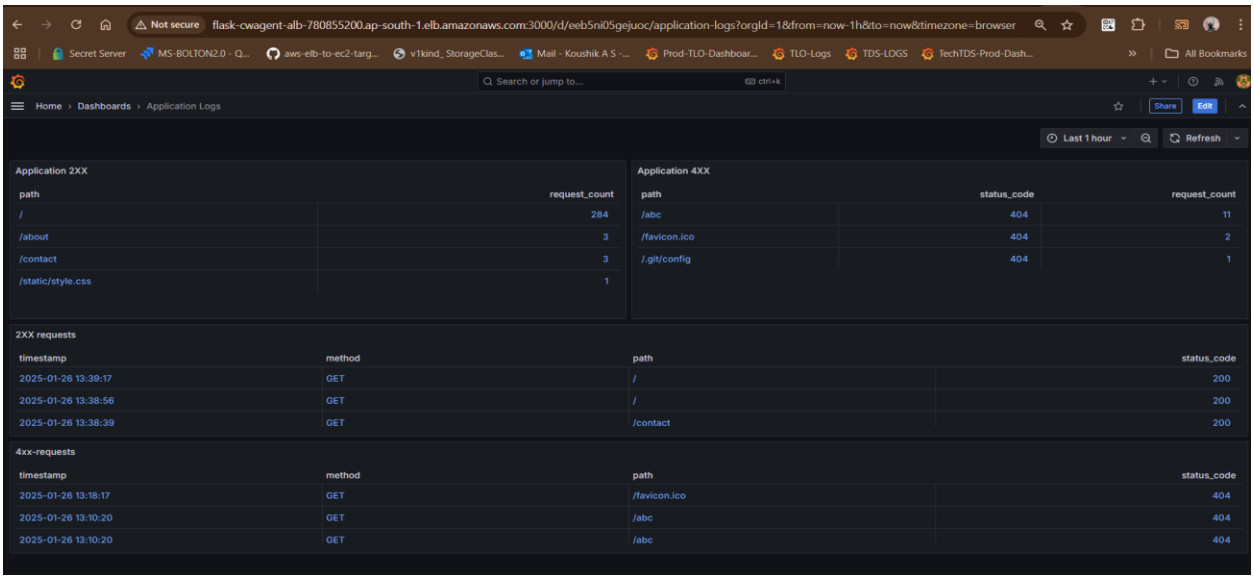
Launch an EC2 instance and install Grafana server for monitoring logs.

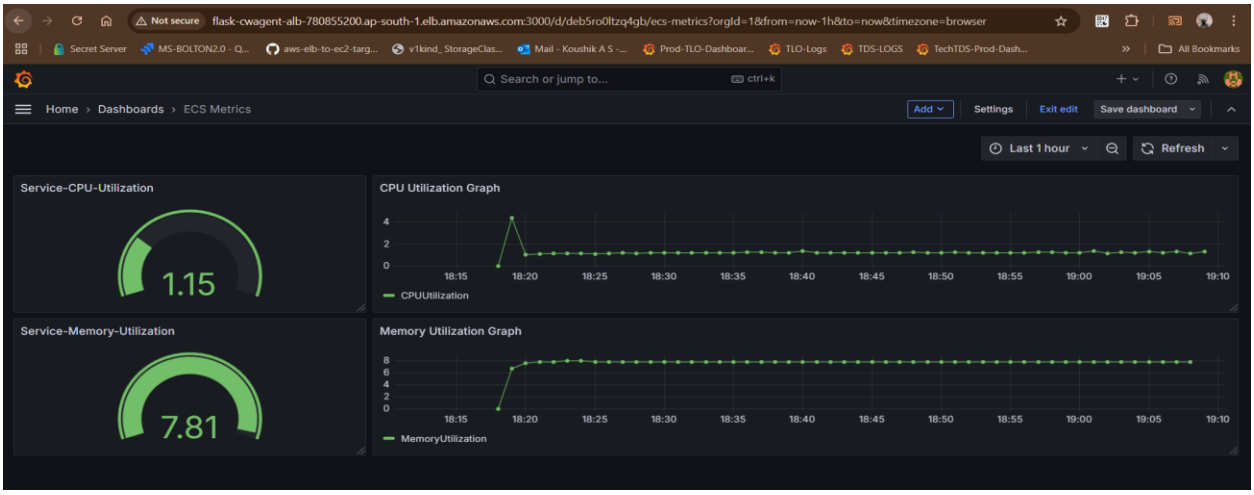


Login and add data source as cloudwatch



Grafana Logs and metrics.





Conclusion:

This architected setup that leverages ECS with Fargate for serverless container management, ALB for traffic distribution, and CloudWatch for logging and monitoring. Grafana provides a powerful visualization layer for analyzing CloudWatch metrics and logs. The integration of a sidecar CloudWatch agent ensures that the Flask application's logs and metrics are collected efficiently, and the Auto Scaling Group helps to automatically scale your resources based on demand, making your application scalable and resilient to changes in load.