ENERGY-CONNECT

About My Application

**Community Energy Connect** is a web-based platform designed to empower communities to adopt renewable energy. It offers a marketplace for eco-friendly products and services, from solar panels to energy audits. Users can explore educational resources, connect with local providers, and make informed sustainability choices. The platform aims to simplify the transition to clean energy through accessibility, awareness, and community engagement.

Why to Use Cloud for this Project?

**Without Cloud (Traditional Hosting):**

* Requires manual setup of servers and network infrastructure
* Difficult to scale when user traffic increases
* Needs in-house expertise for security, backup, and maintenance
* Slower deployment and updates (manual CI/CD)
* Upfront hardware and infrastructure costs

**With Cloud (AWS/GCP/Azure, etc.):**

* **On-demand infrastructure** – Instantly provision servers, storage, databases
* **CI/CD with CodeBuild & CodeDeploy** – Automate testing, building, and deployment
* **Reliable networking** – Easily manage traffic, security groups, load balancing
* **Scalability** – Auto-scale based on user demand without downtime
* **Built-in security & monitoring** – IAM, CloudWatch, backups, DDoS protection
* **Cost-efficient** – Pay only for what you use, no upfront hardware needed
* **Global availability** – Deploy apps close to users with low latency

**INFRA FOR THE PROJECT:**

**AWS CloudFormation:**

CloudFormation is an AWS service that lets you define your entire infrastructure as code (IaC) using a YAML or JSON template.

**How It Works – Basic Flow**

1. You write a template (YAML/JSON)

↓

2. You submit it to CloudFormation (as a "stack")

↓

3. CloudFormation provisions all resources

↓

4. You can update, delete, or manage the stack later

* **VPC (Virtual Private Cloud):**  
  Creates an isolated network environment in the cloud.
* **Public Subnets:**  
  Used to host resources that must be accessible from the internet.
* **Private Subnets:**  
  Host backend services like databases or internal applications that shouldn’t be exposed to the internet.
* **Internet Gateway (IGW):**  
  Allows public subnets to communicate with the internet.
* **NAT Gateway:**  
  Enables resources in private subnets to initiate outbound internet while still blocking unsolicited inbound traffic.
* **Route Tables:**  
  Control the traffic flow between subnets and to/from

* **Availability Zones (AZs):**  
  Distribute resources across multiple data centers to ensure high availability and fault tolerance.
* **EKS**
* **ECR**
* **RDS**

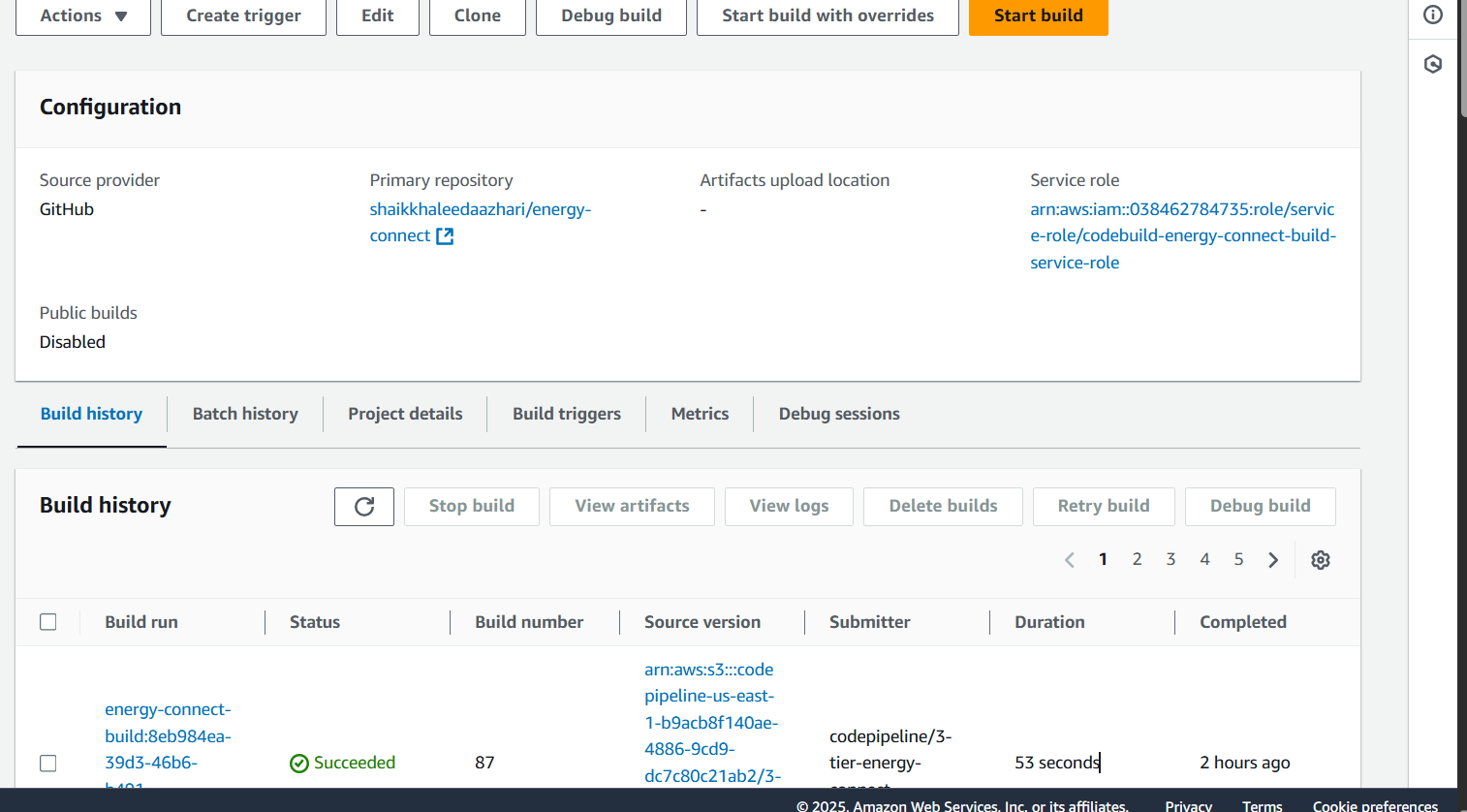
Applications Building and Deployment:

AWS CodeBuild:

AWS **CodeBuild** is a fully managed CI service that compiles source code, runs tests, and produces software packages that are ready to deploy..

**CodeBuild Workflow**

1. **Source** (e.g., GitHub, CodeCommit, S3)
2. **CodeBuild** picks up source → uses buildspec.yml
3. **Build process** is executed (e.g., install dependencies, run tests, build artifacts)
4. **Artifacts** (e.g., Docker images, zip files) are stored (S3, ECR)



For the Project:

* I created two Docker files one for frontend and other for backend.
* So two images will be built by the buildspec
* Built is nothing but the installing, adding necessary addons.
* Then it will create an artifact which is our docker images.
* We will then push into the ECR.

**ECR:**

Elastic Container Registry, which will store the images, like docker hub.

Poweruser role to push image after building to ecr

AmazonEC2ContainerRegistryPowerUser

**Amazon EKS**

**EKS runs Kubernetes for you** — so you don’t have to install, manage, or scale the control plane and other core components.

**EKS Workflow**

You define app in YAML (Deployment, Service, etc.)

↓

kubectl apply

↓

EKS schedules pods on worker nodes

↓

Pods run containers (Docker, etc.)

↓

EKS manages availability, networking, and scaling

**AWS CodePipeline**

CodePipeline is a fully managed CI/CD service in AWS that automates the build, test, and deploy phases of your software release process.

**CodePipeline Flow Example**

1. Developer pushes code to CodeCommit/GitHub

↓

2. CodePipeline is triggered

↓

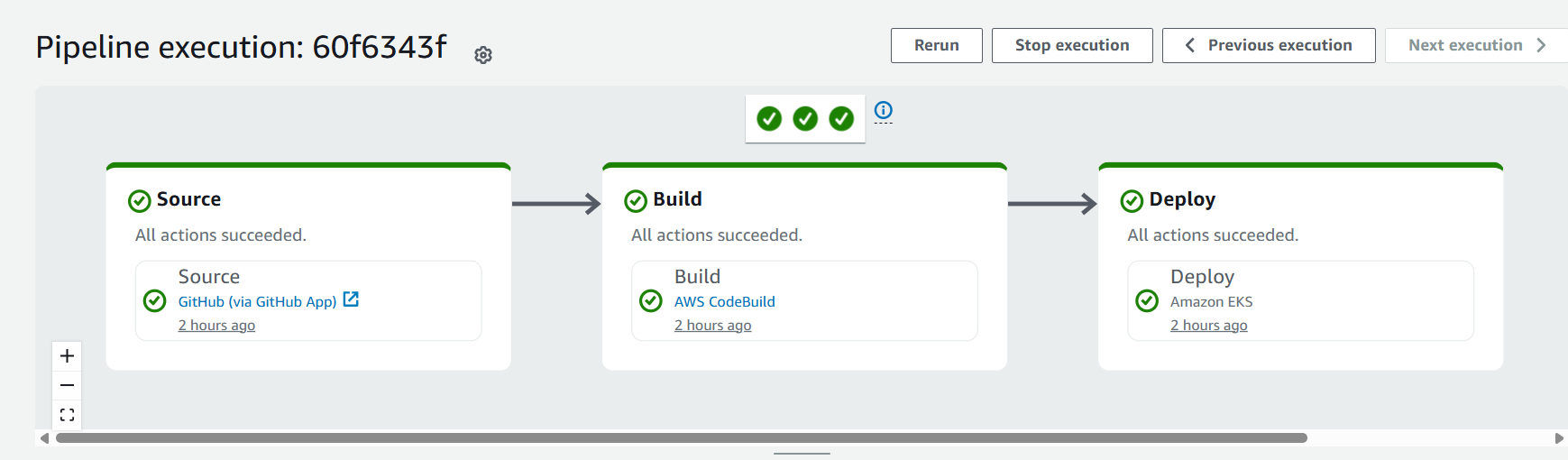
3. CodeBuild compiles & tests code (via buildspec.yml)

↓

4. Artifacts pushed to S3 or Docker image to ECR

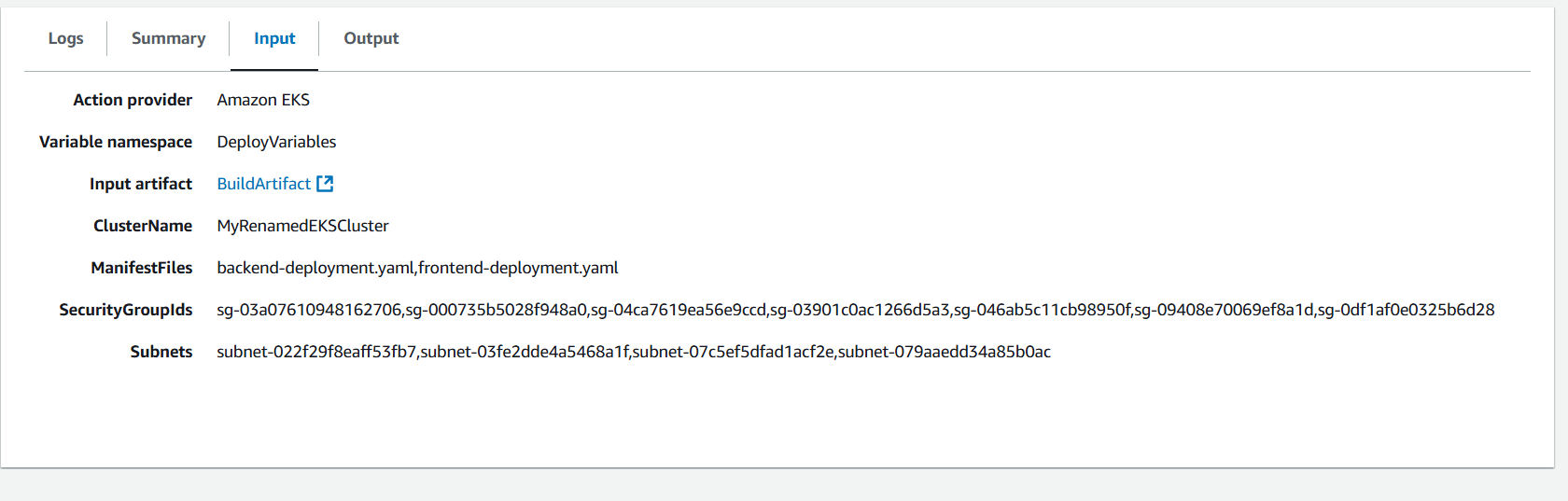
↓

5. CodeDeploy deploys to EC2, ECS, Lambda, or EKS

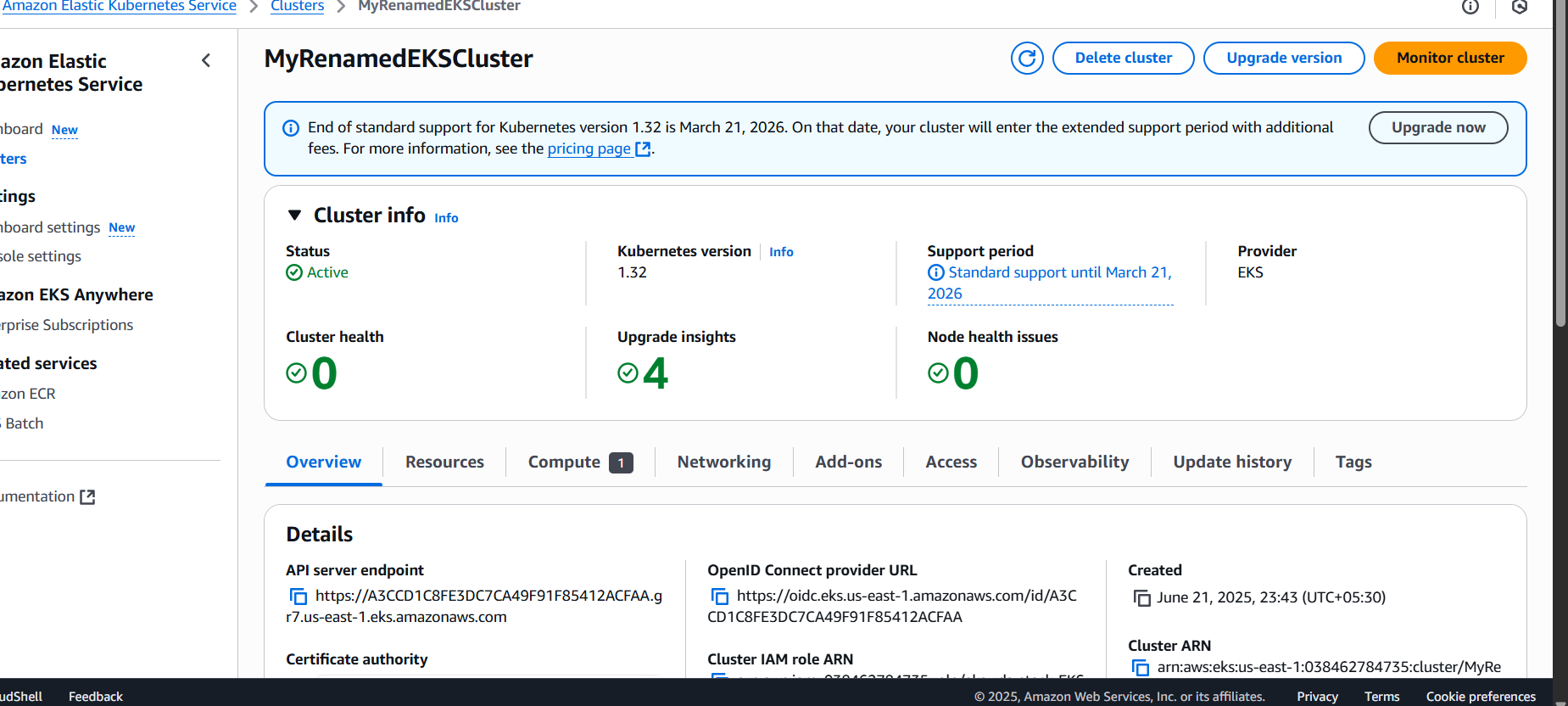


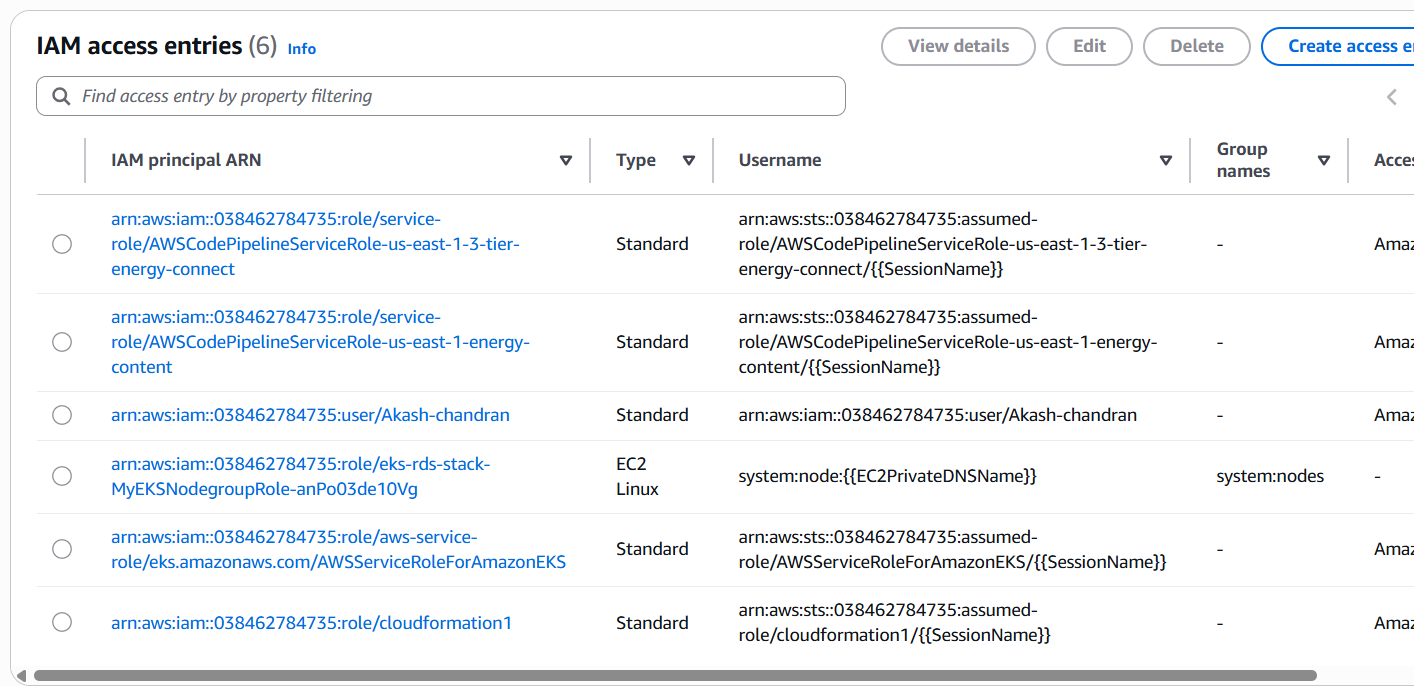
Deploy:

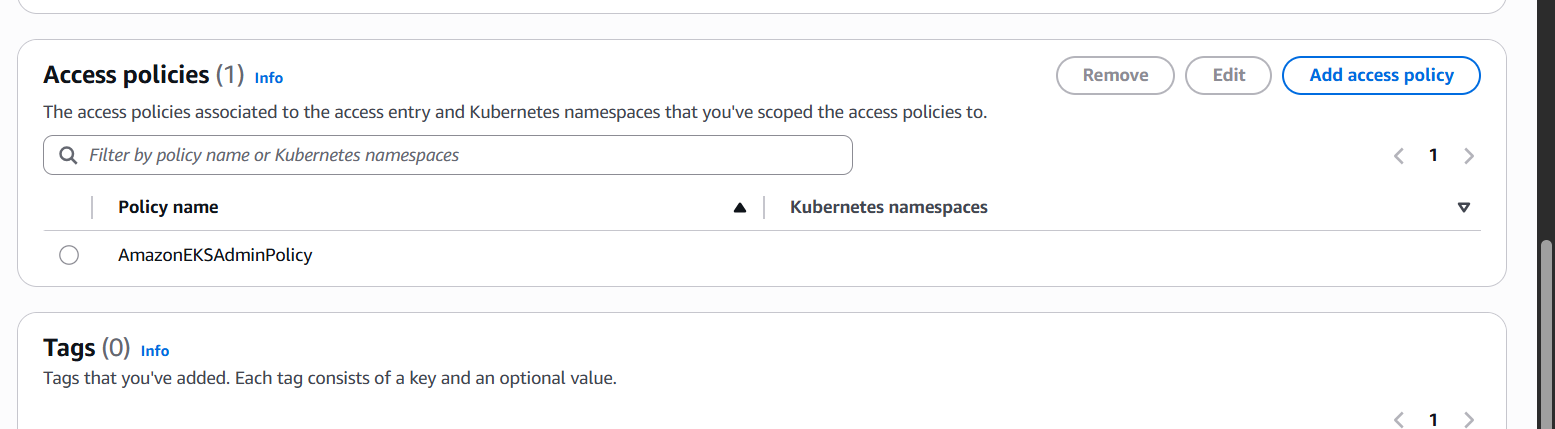
* We can deploy in Ec2, Ecs, lambda, on-premises servers



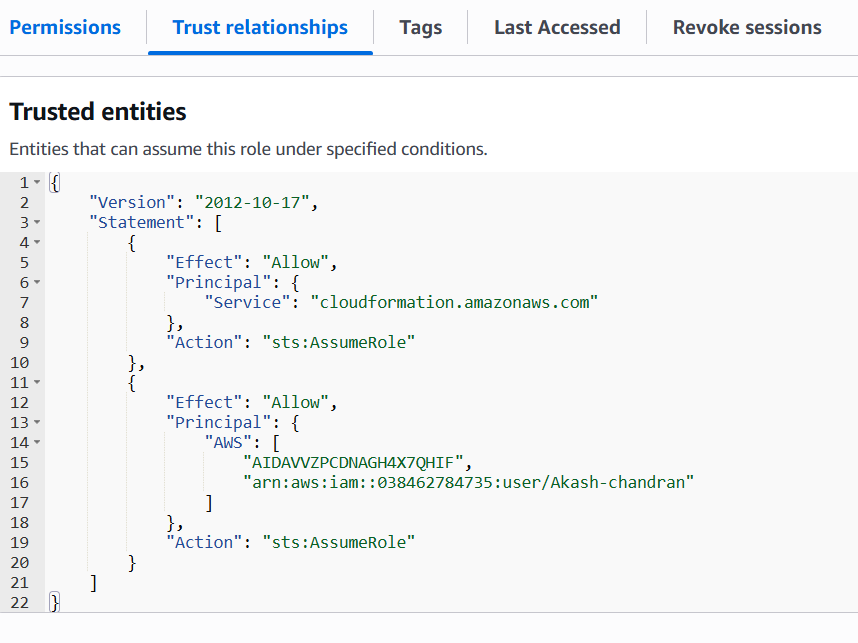
* Deploy will take the Build Artifact and deploy the applications in to eks.



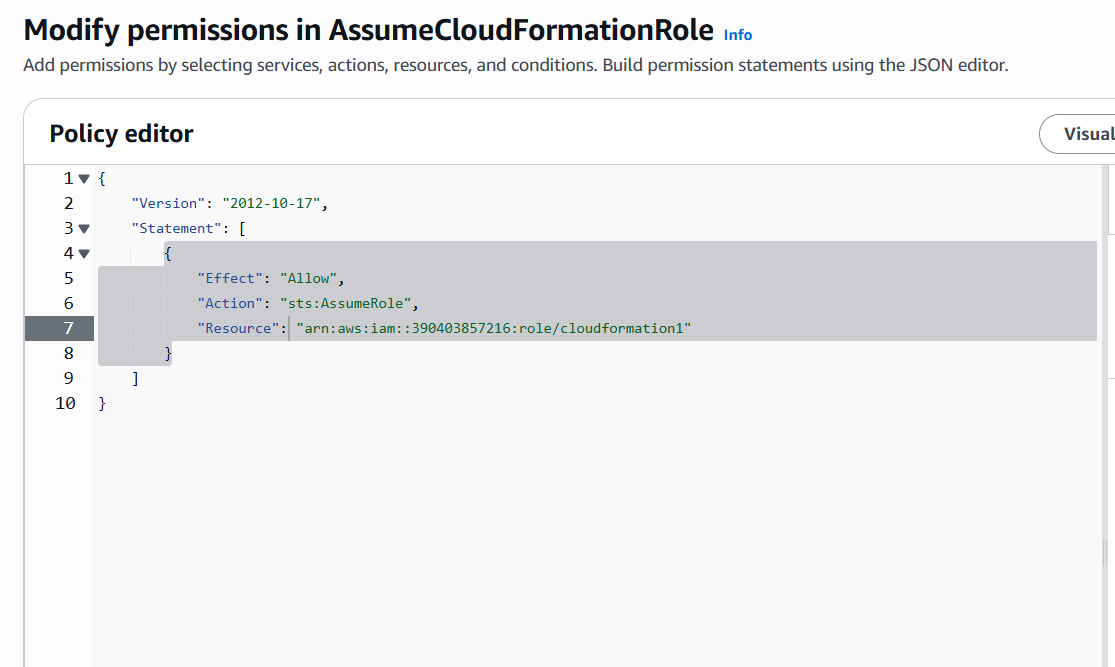




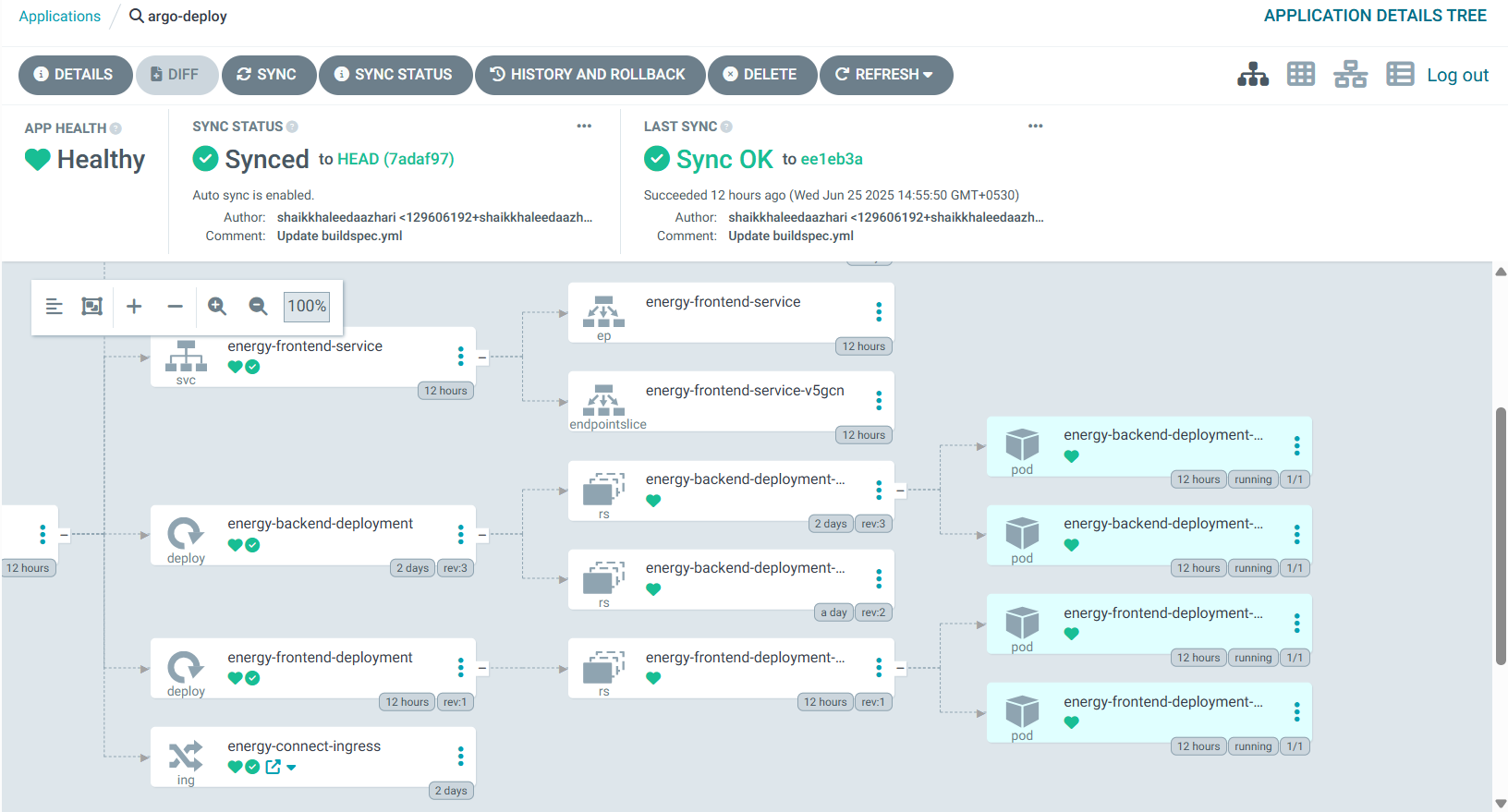
Add the user in Trust relationships

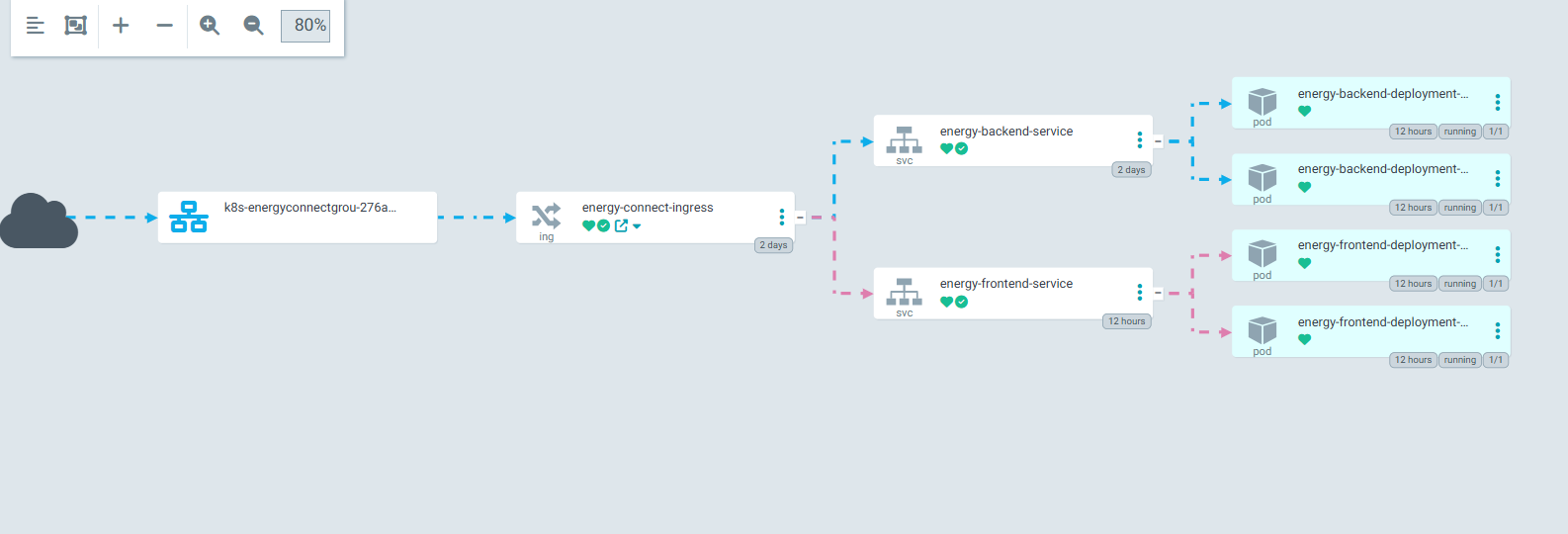


Add cloud watch permissions to the user



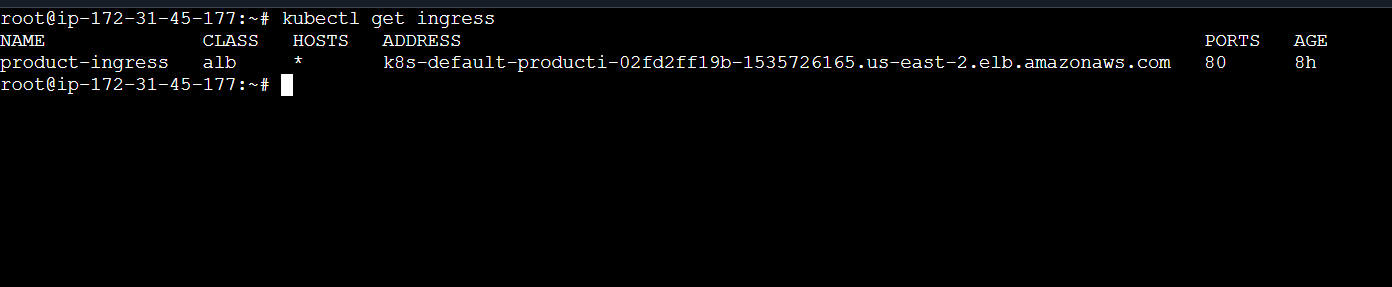
Argo CD (Argo Continuous Delivery) is a GitOps-based Continuous Delivery tool for Kubernetes. It automates application deployment and lifecycle management using Git as the single source of truth.



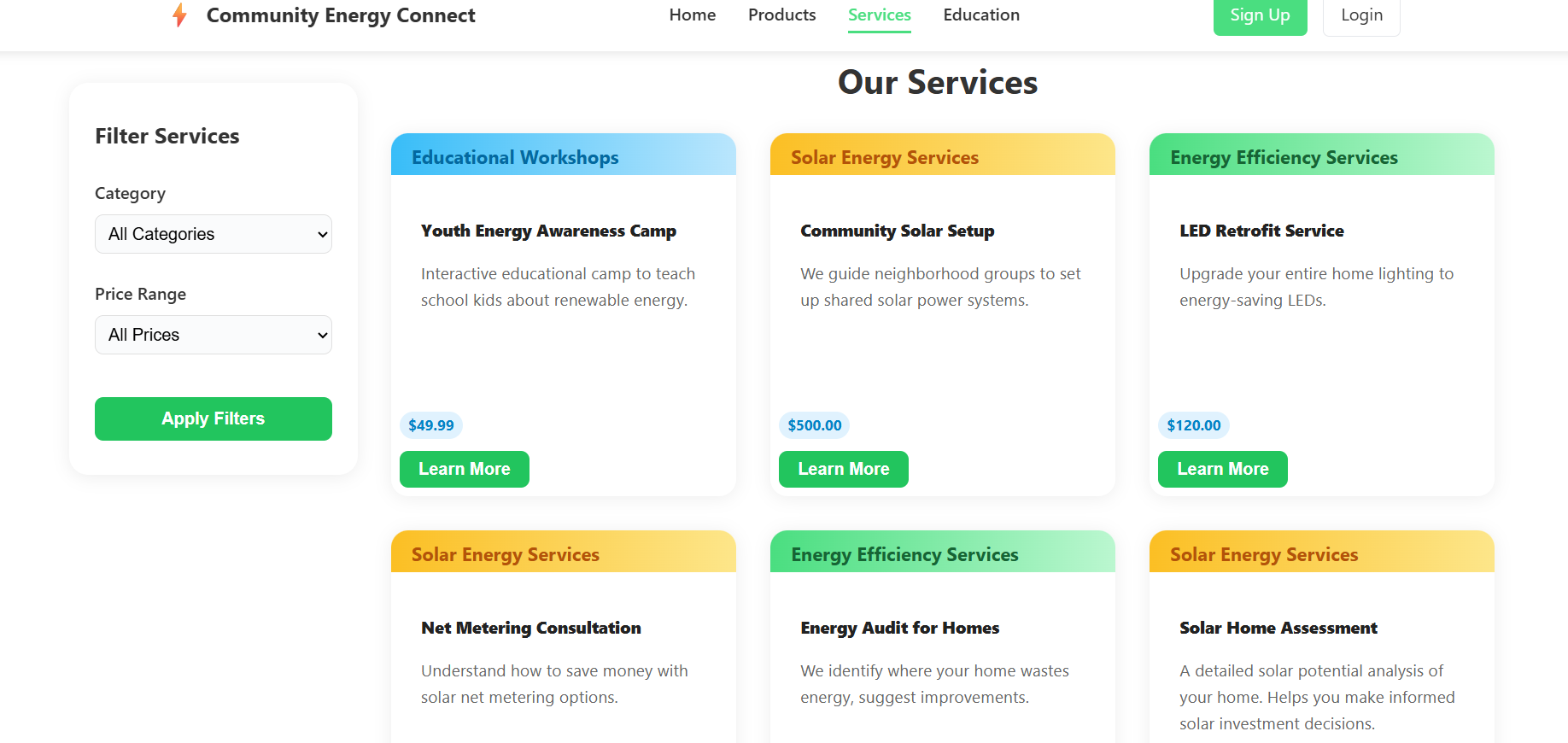


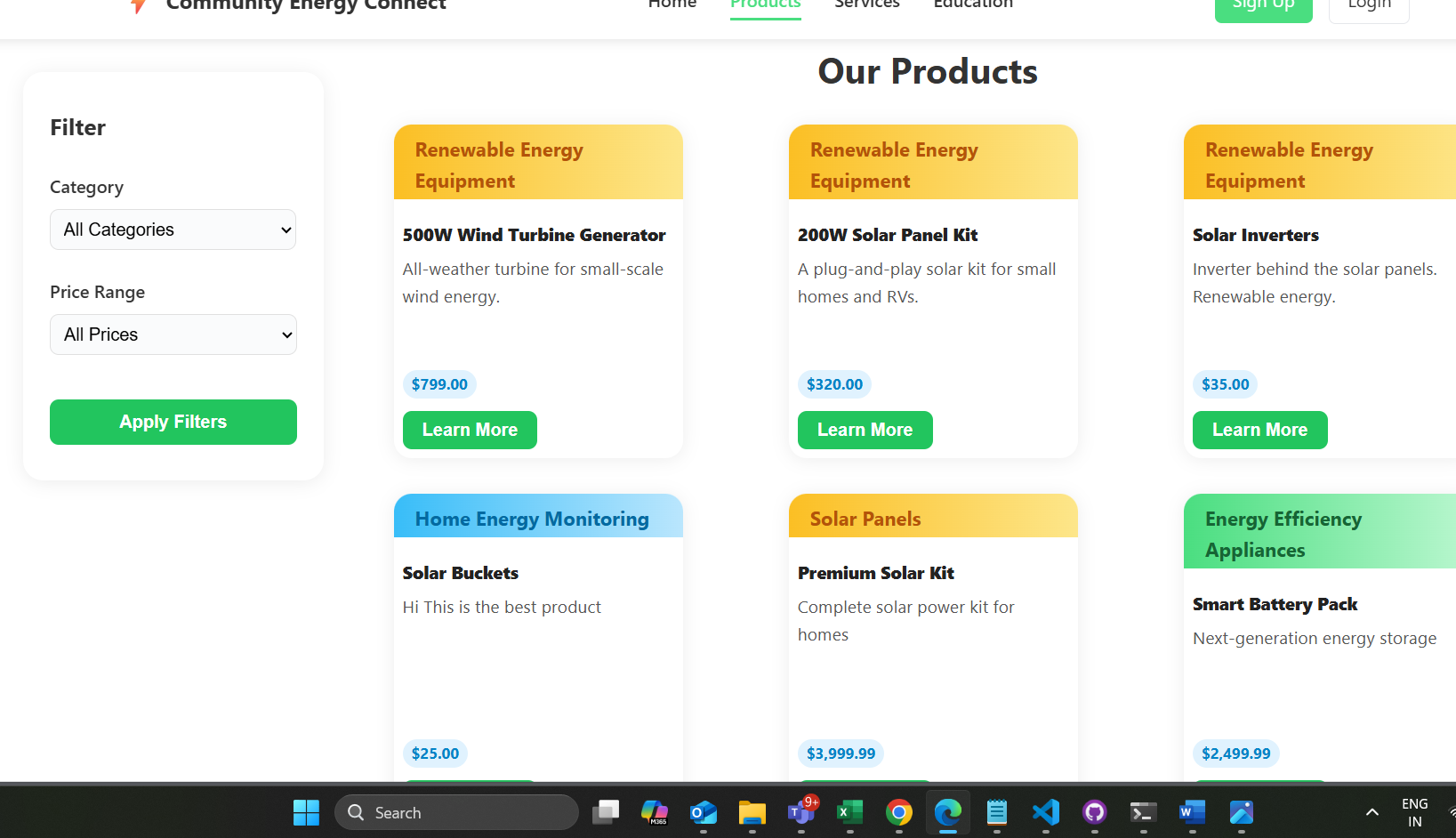
After the deployment to give the internet facing access

So that users can access.





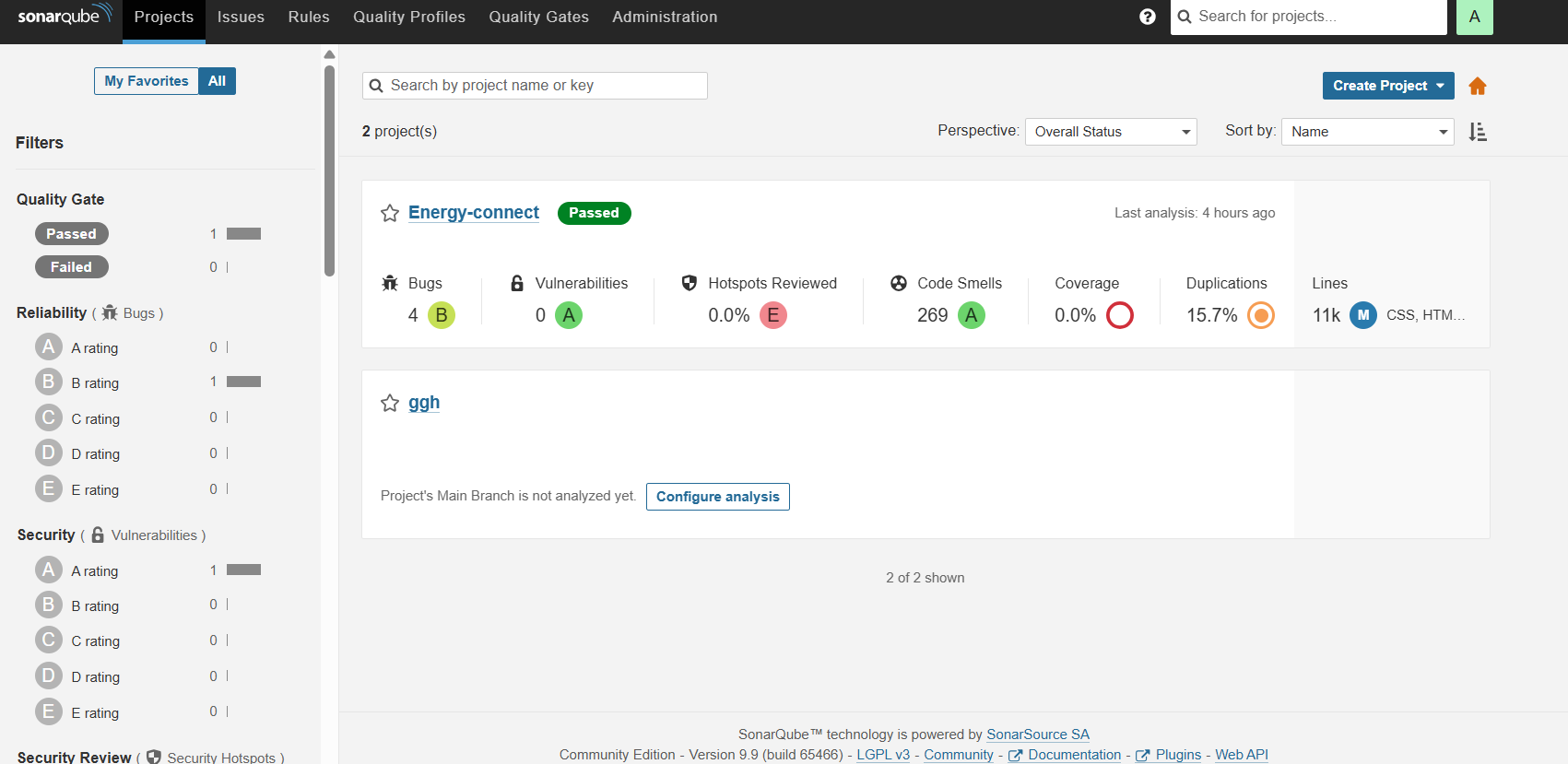


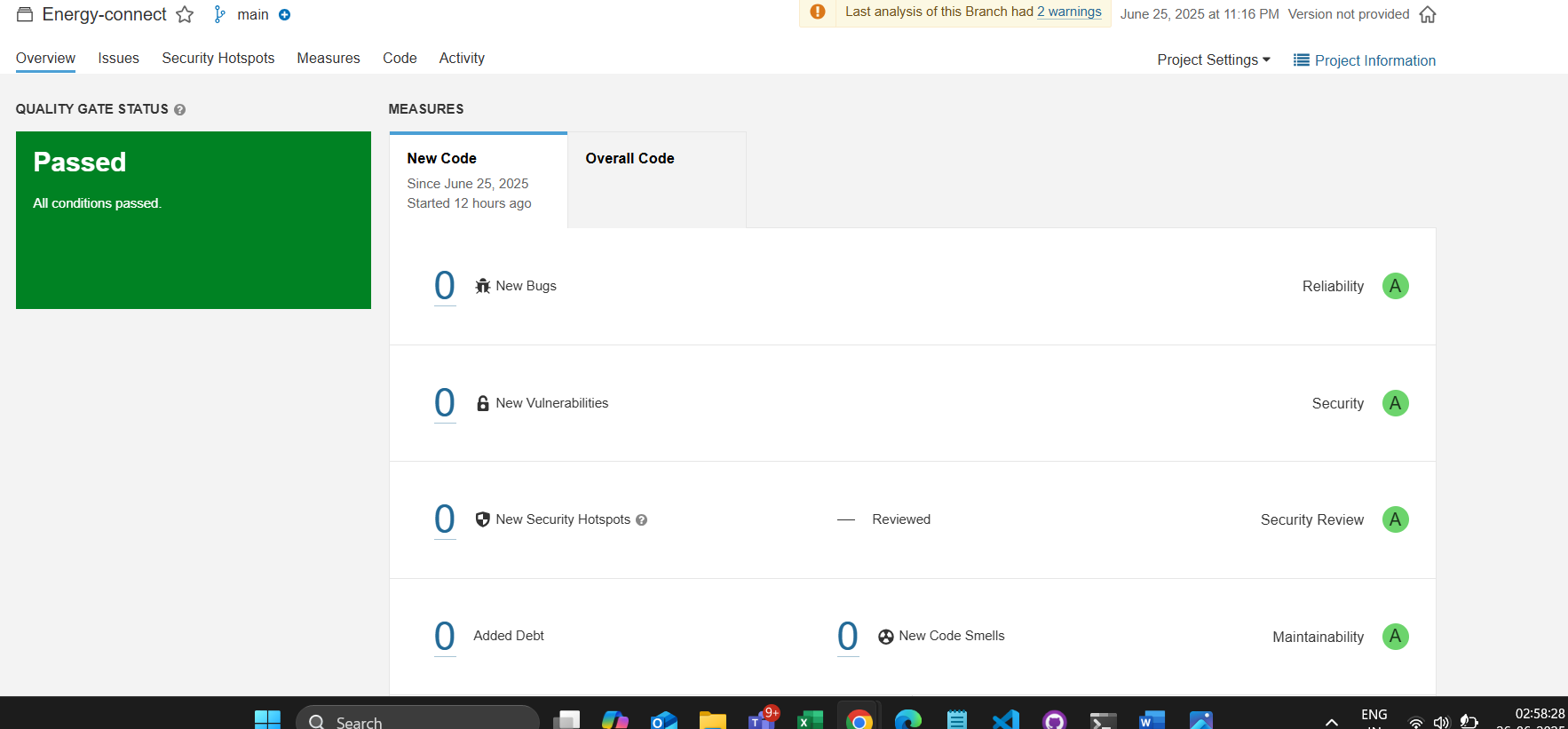


**SonarQube:**

SonarQube is a powerful open-source tool used to inspect, analyze, and improve code quality automatically. It supports continuous inspection of codebases to detect

* Bugs
* Code smells
* Security vulnerabilities
* Technical debt
* Duplicated code

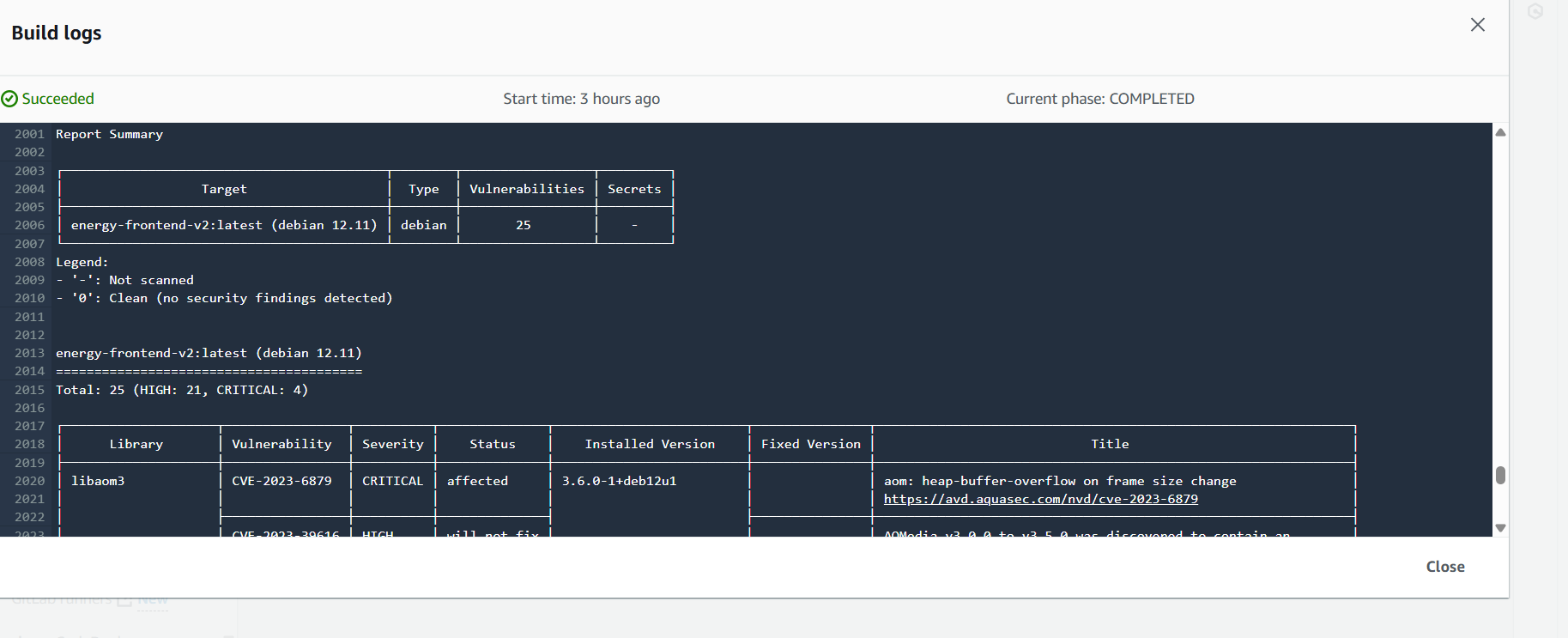




TRIVY:

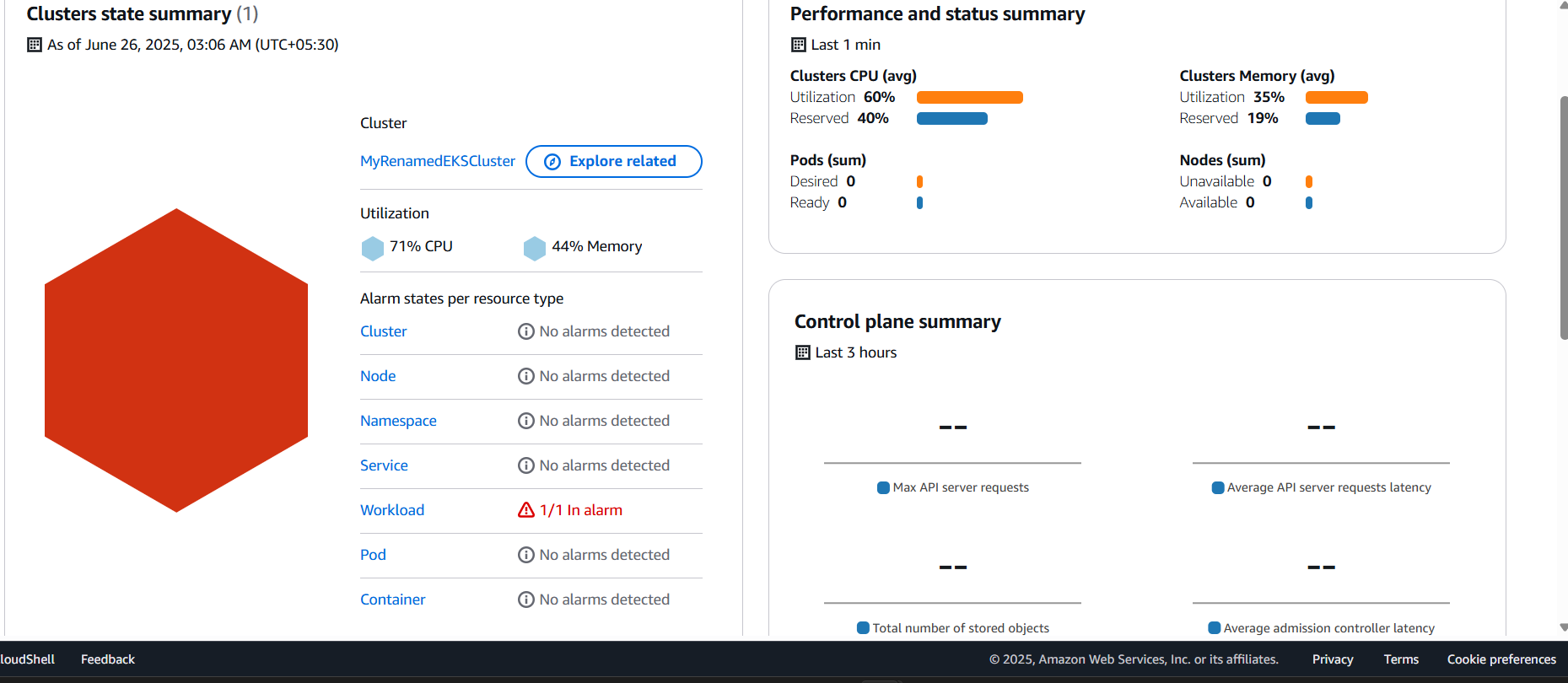
Trivy (pronounced *triv-ee*) is a simple and comprehensive security scanner for:

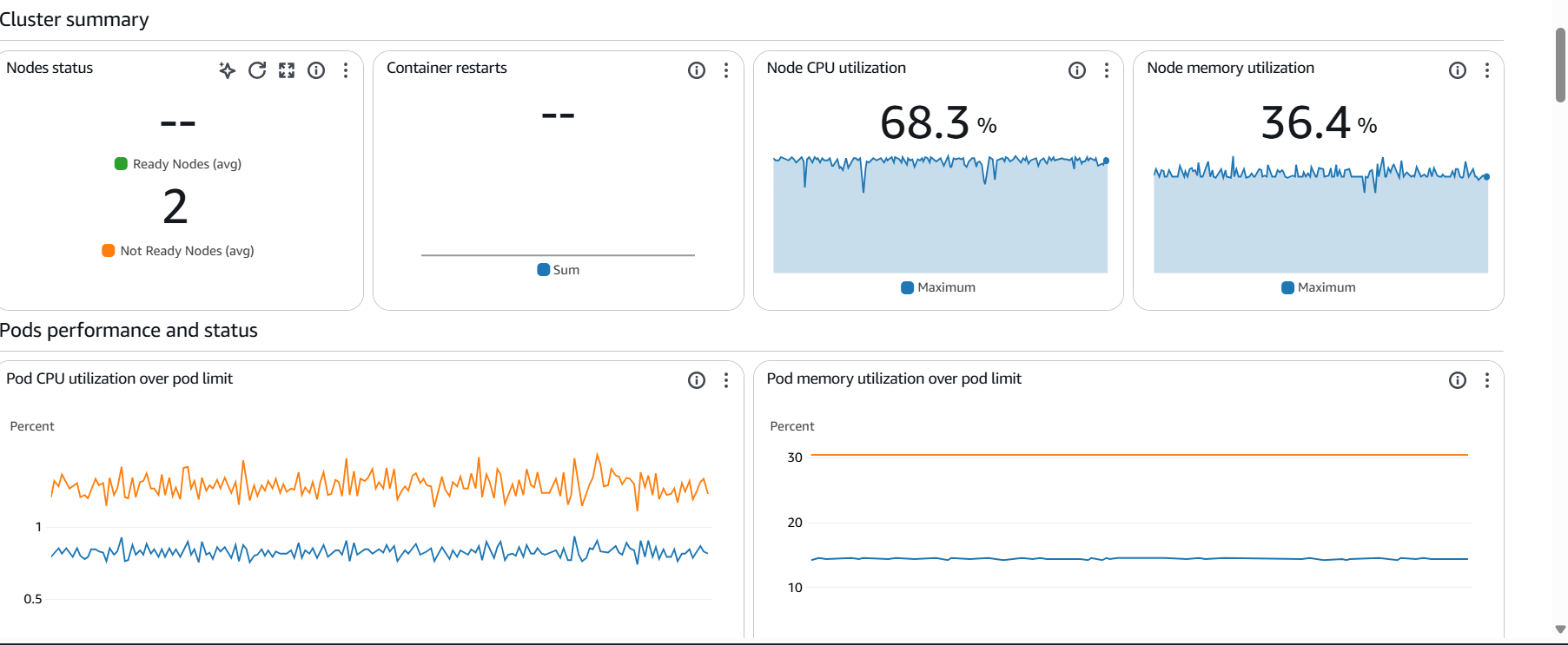
* Container images
* File systems
* Git repositories
* Kubernetes clusters

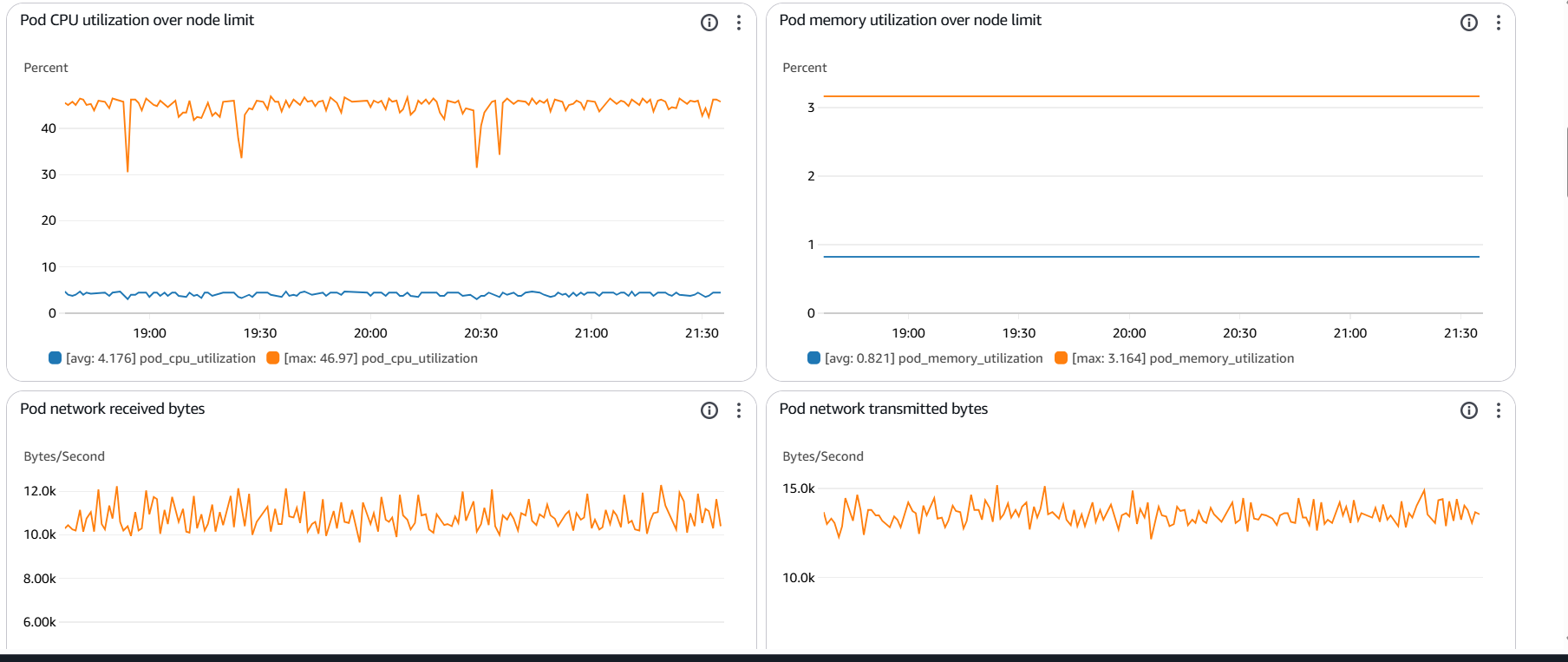


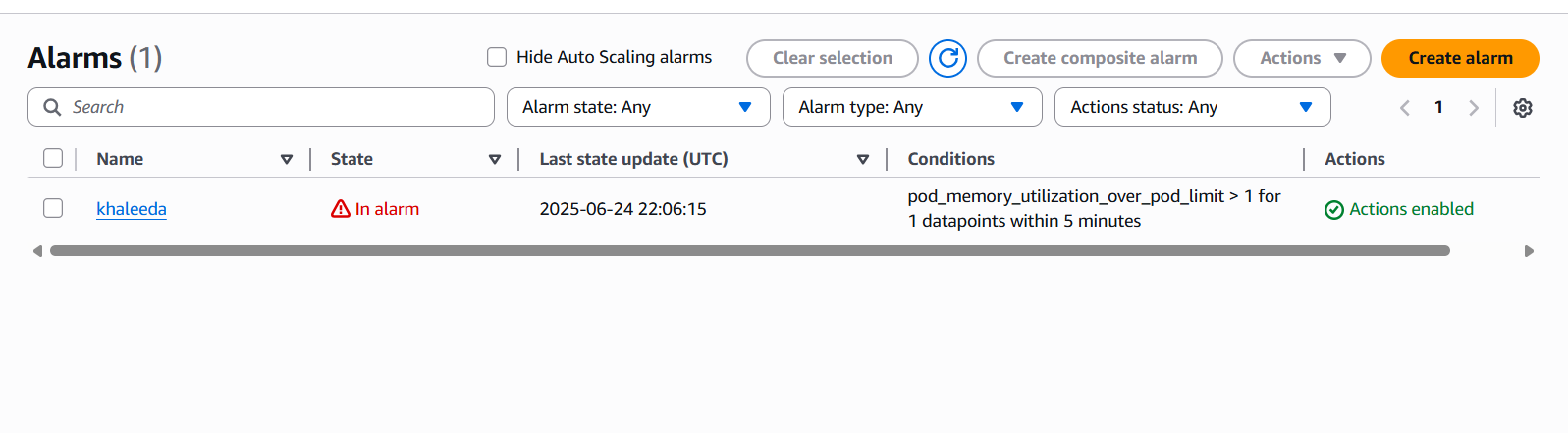
Cloud Watch:

**Amazon CloudWatch** is a **monitoring and observability** service provided by AWS. It collects **metrics, logs, and events** from your AWS resources, applications, and services, and allows you to monitor, analyze, and react to them in near real time.









Route 53:

Amazon Route 53 is AWS’s Domain Name System (DNS) web service

