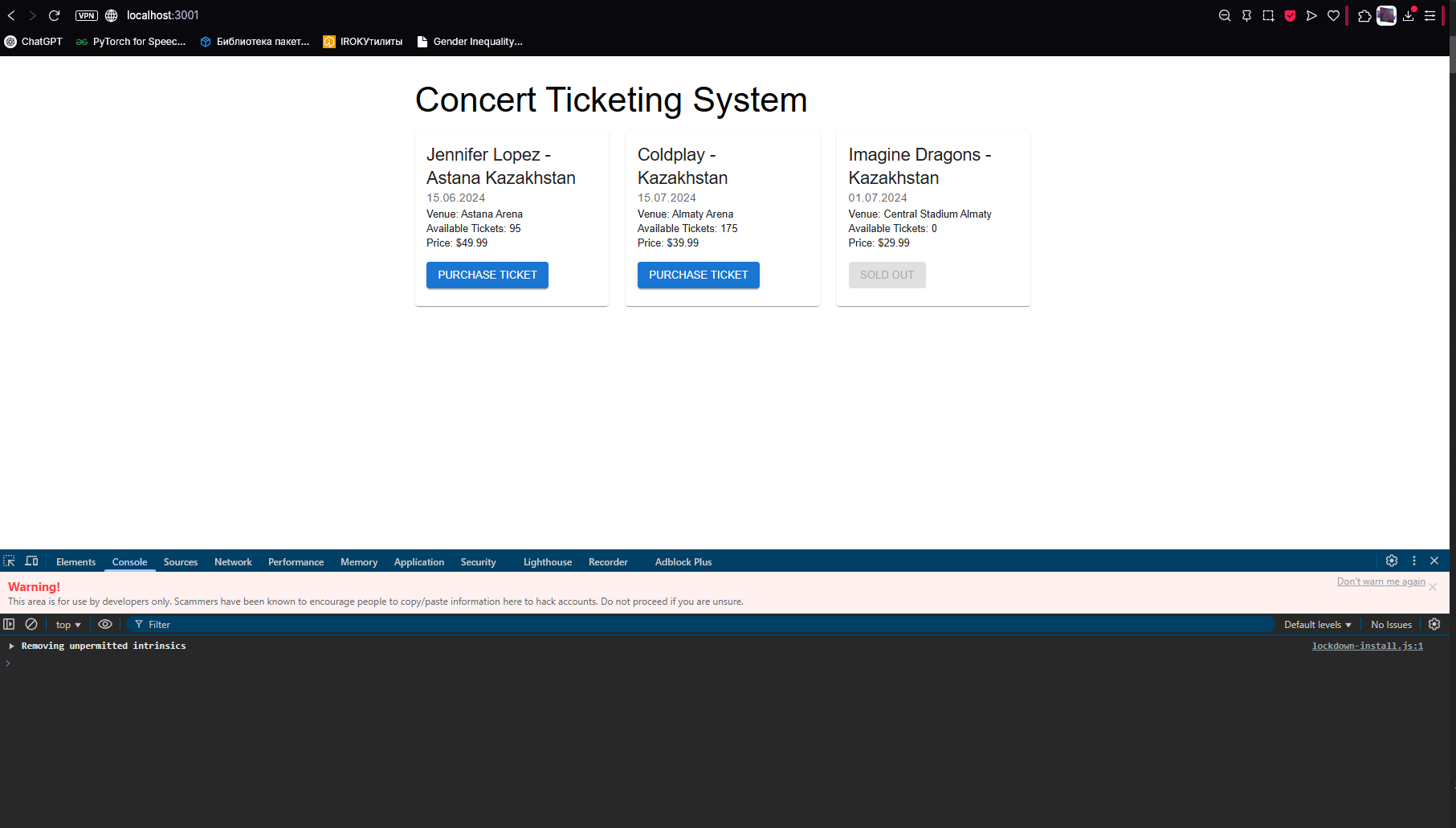
SREFINALZHAS:

TASK 1

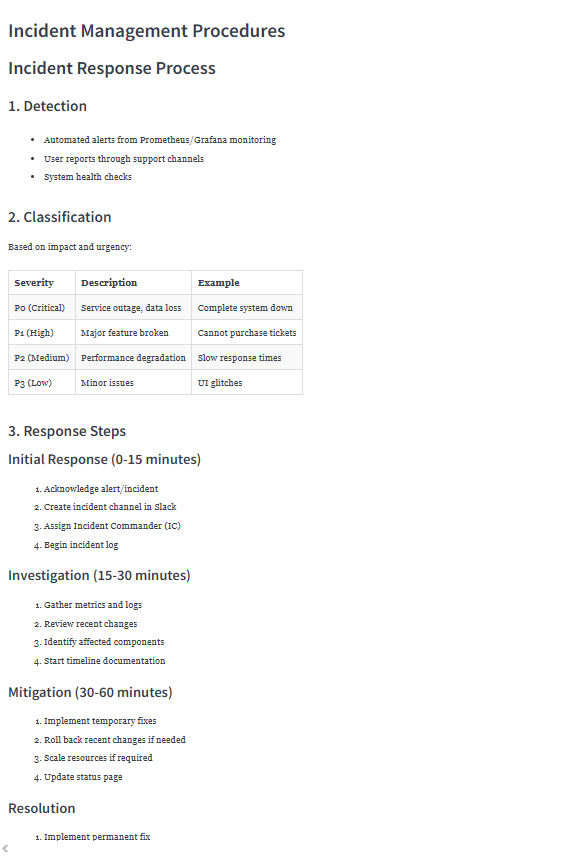
1) created a concert ticketing web application

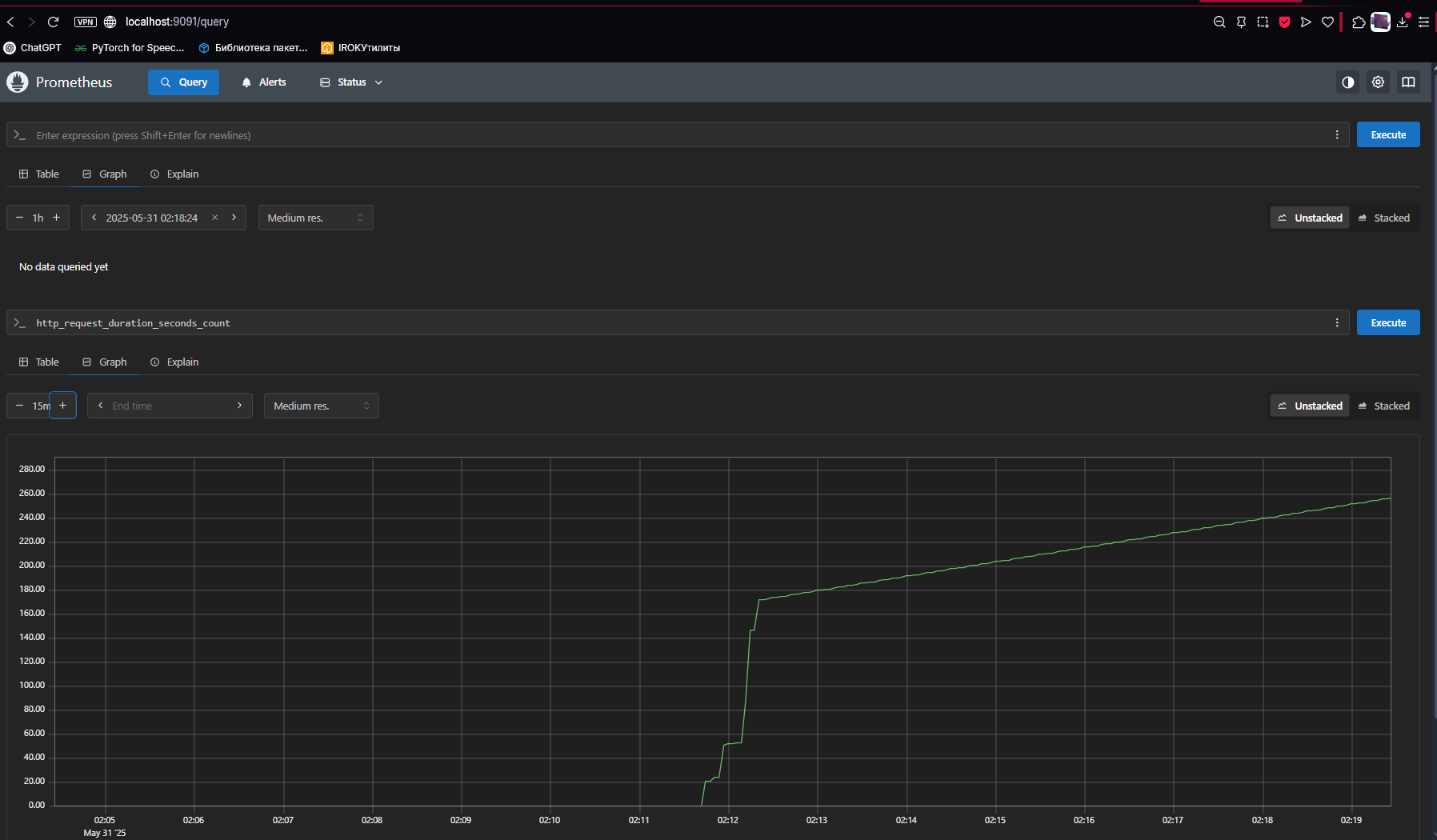
Components: React frontend, FastAPI backend, PostgreSQL database (RDS), Containerized with Docker, Deployed on AWS ECS Fargate

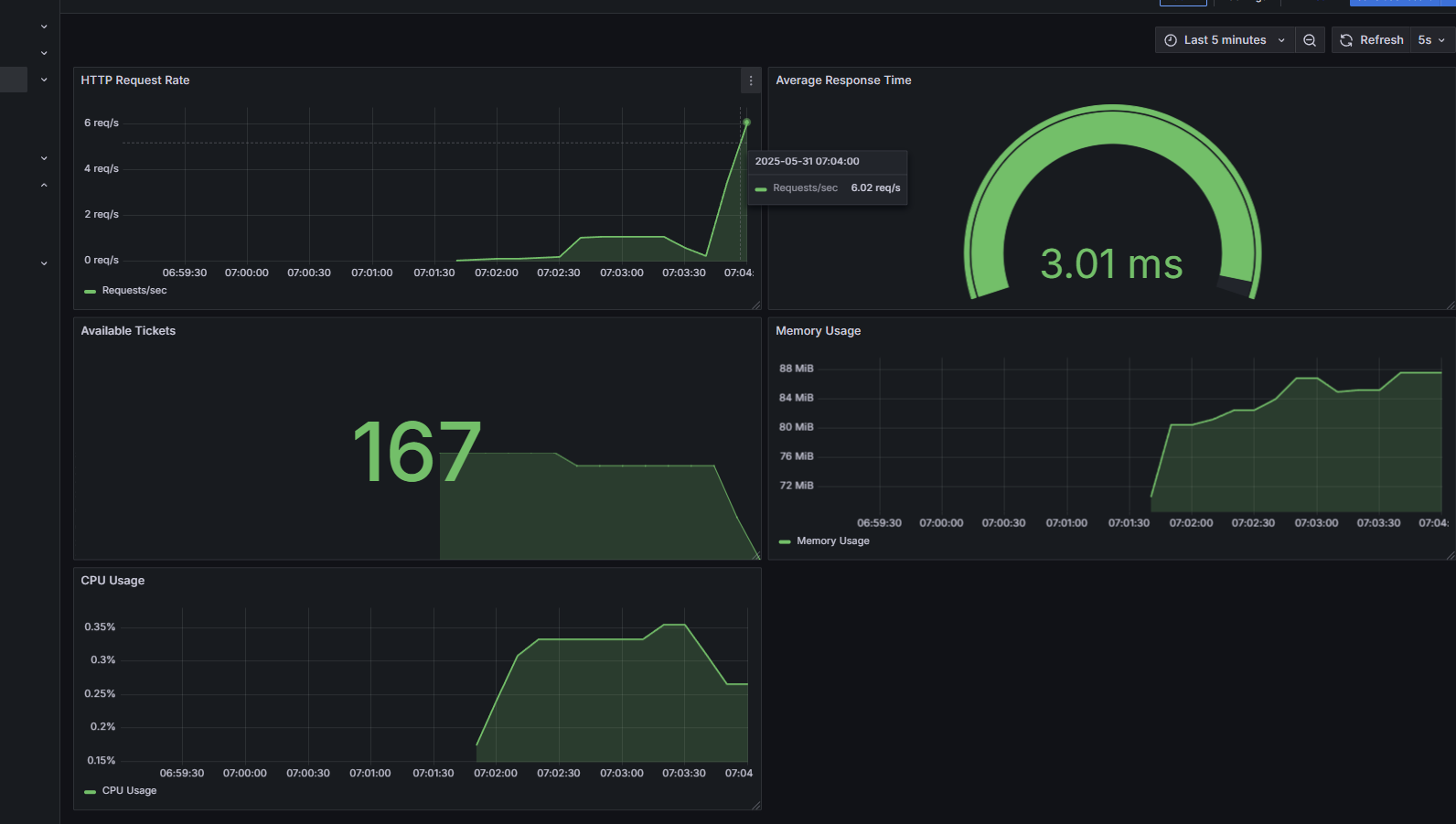


2) Monitoring & Alerting:

Comprehensive SRE Documentation, Detailed SLI/SLO definitions, SLA with service commitments, Incident management procedures, Postmortem template, Error budget policies







Defined key metrics:

HTTP Request Rate

Average Response Time

Available Tickets

CPU Usage

Memory Usage

**Task 2 - Infrastructure as Code Collaboration**

Application & Infrastructure Design:

Multi-container architecture using Docker Compose

Services: Frontend, Backend, Database, Prometheus, Grafana

High availability considerations through container orchestration

IaC Implementation:

Used Docker Compose for infrastructure definition

Services are containerized and easily deployable

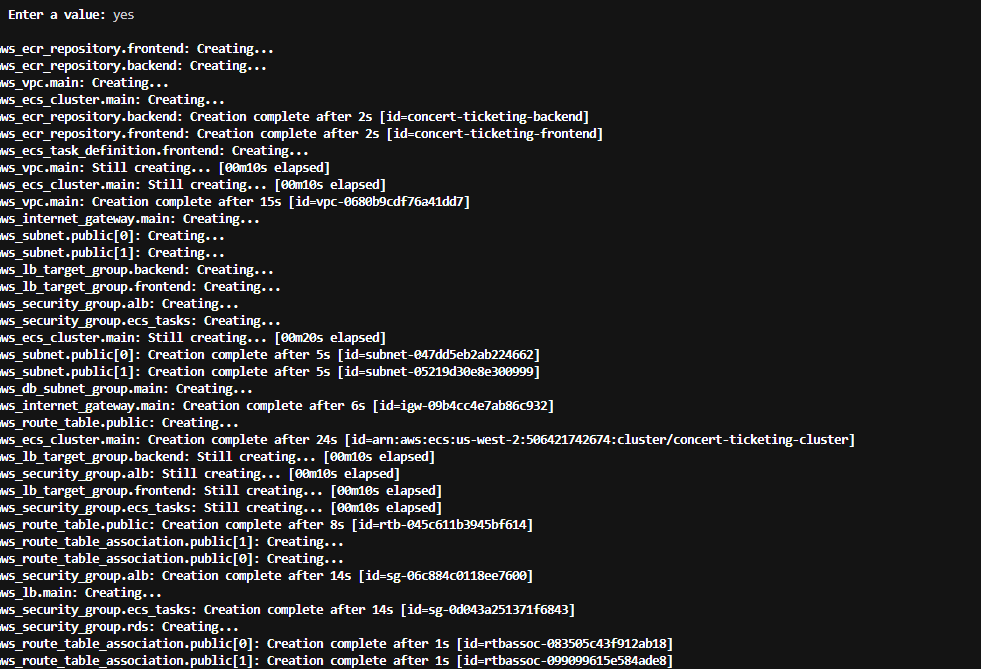
**AWS Infrastructure (Terraform):**

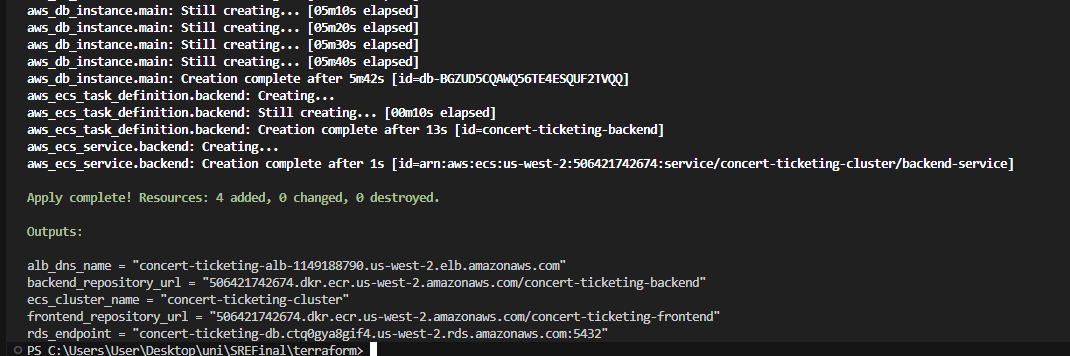
* VPC with public subnets across multiple AZs
* ECS Fargate cluster for container orchestration
* ECR repositories for Docker images
* RDS PostgreSQL database
* Application Load Balancer
* Security groups and networking

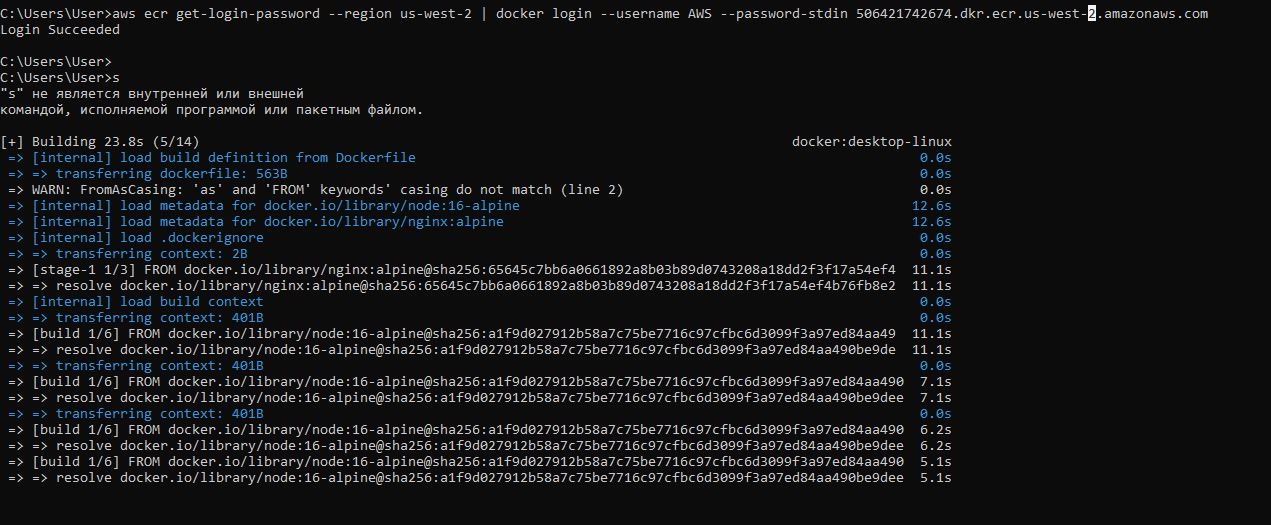
Security Considerations:

* Proper security groups configured
* IAM roles and policies implemented
* Network isolation with VPC
* Secure database access

Terraform log:







Outputs:

alb\_dns\_name = "concert-ticketing-alb-1149188790.us-west-2.elb.amazonaws.com"

backend\_repository\_url = "506421742674.dkr.ecr.us-west-2.amazonaws.com/concert-ticketing-backend"

ecs\_cluster\_name = "concert-ticketing-cluster"

frontend\_repository\_url = "506421742674.dkr.ecr.us-west-2.amazonaws.com/concert-ticketing-frontend"

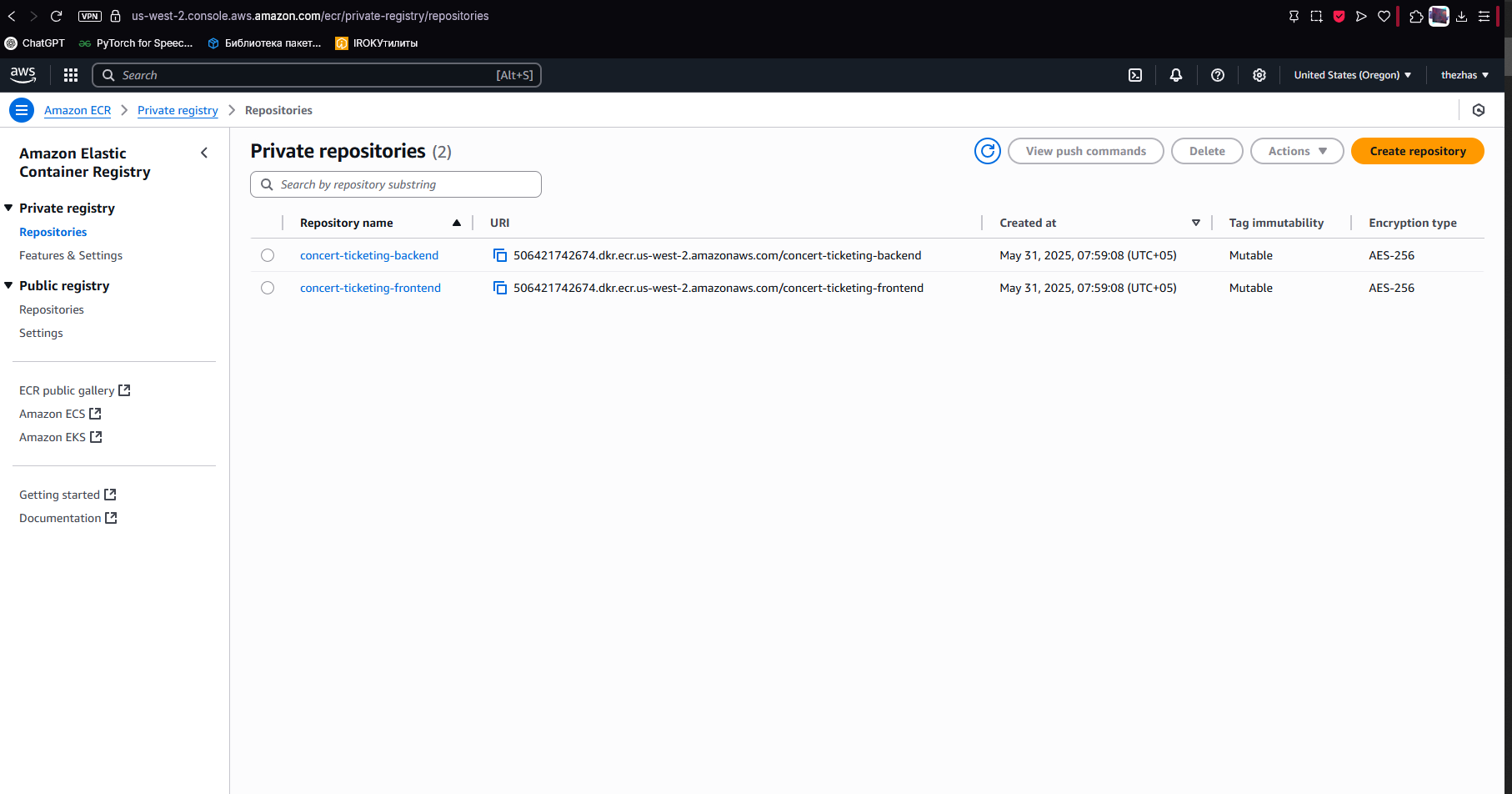
rds\_endpoint = "concert-ticketing-db.ctq0gya8gif4.us-west-2.rds.amazonaws.com:5432"

Integration & Deployment:

All services are integrated and work together

Environment variables and networking properly configured

Volumes for data persistence

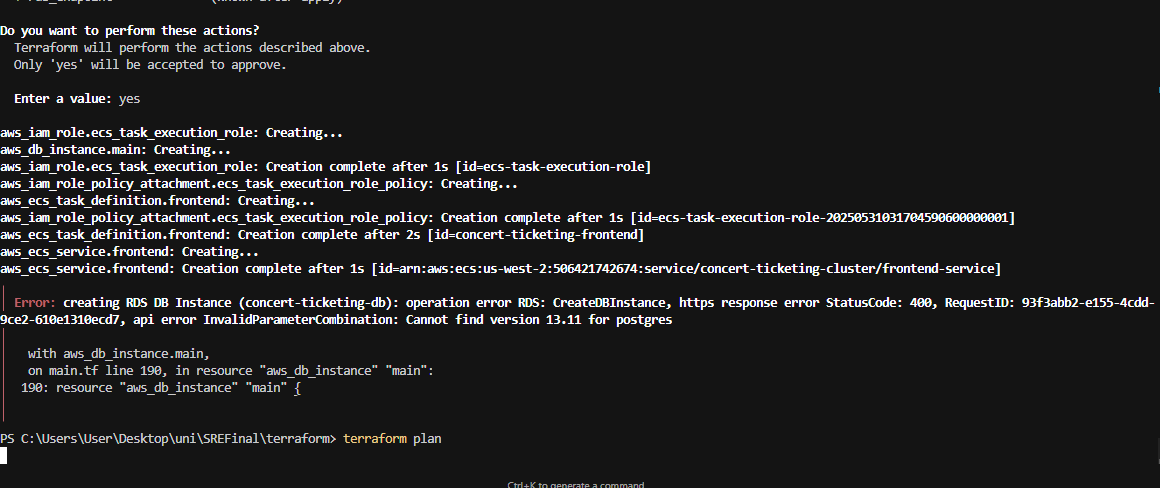


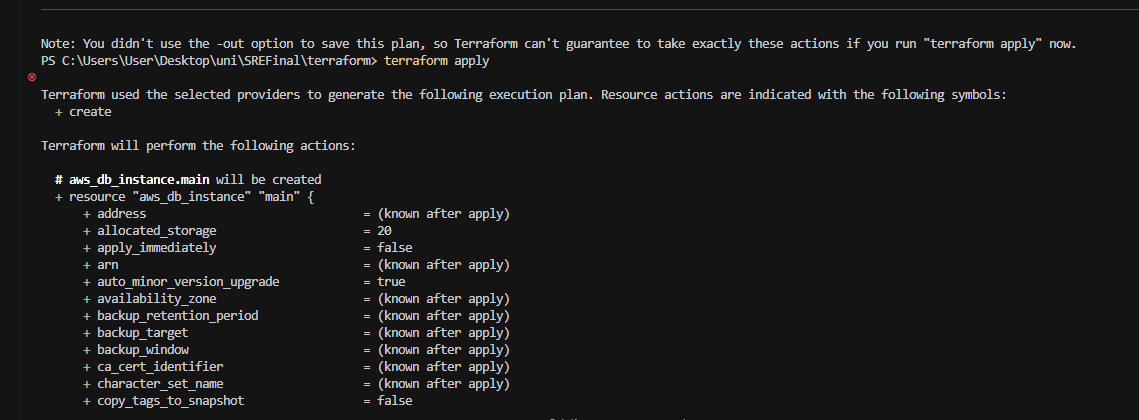
Documentation:

Used best practices

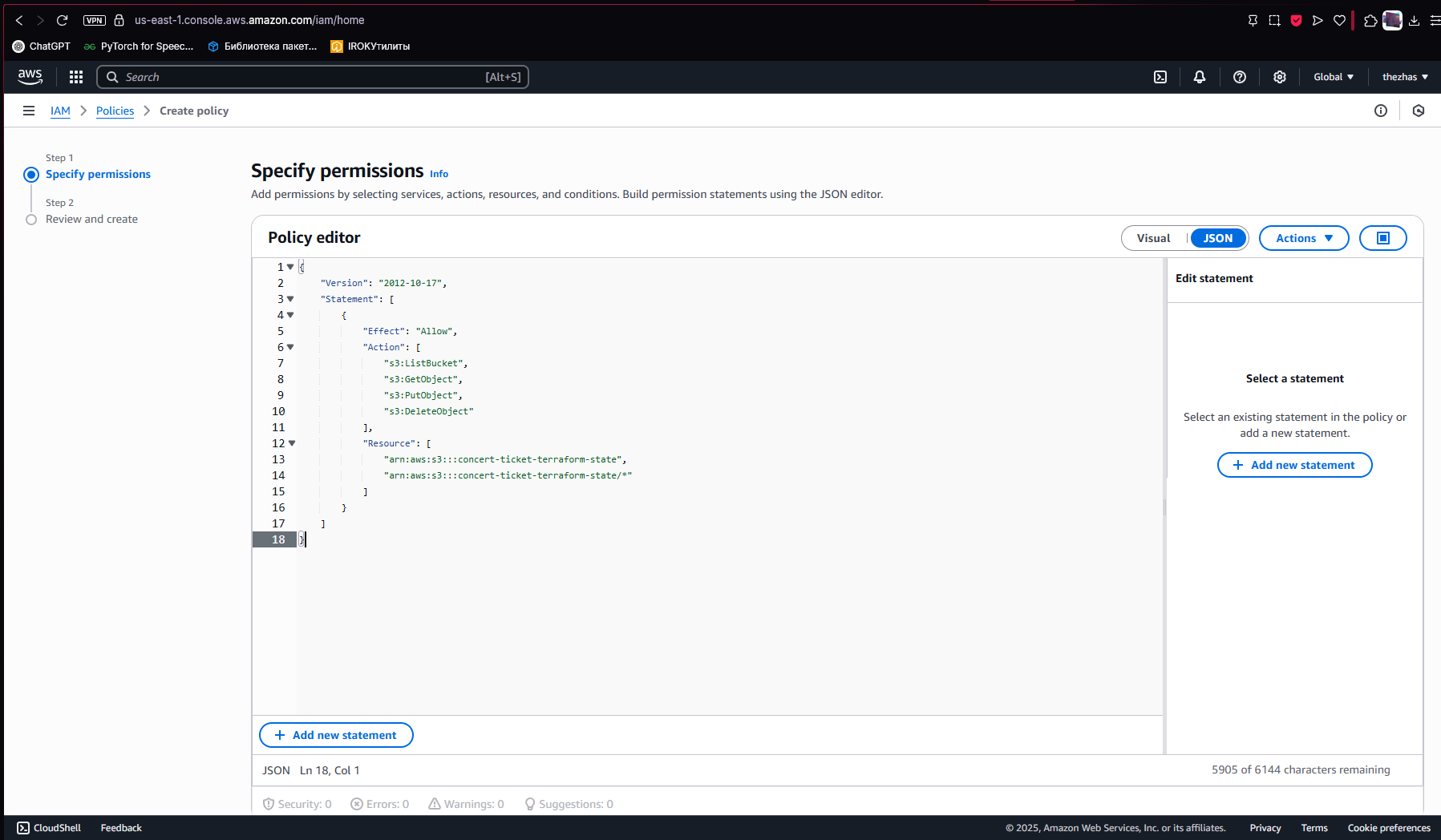
Problems and challenges faced: CORS problems between frontend and backend. Fixed it creating provider for Grafana.  
  
Counter dec() function for tickets error. Fixed this by using a Gauge instead of a Counter for the available tickets metric since it can both increase and decrease:

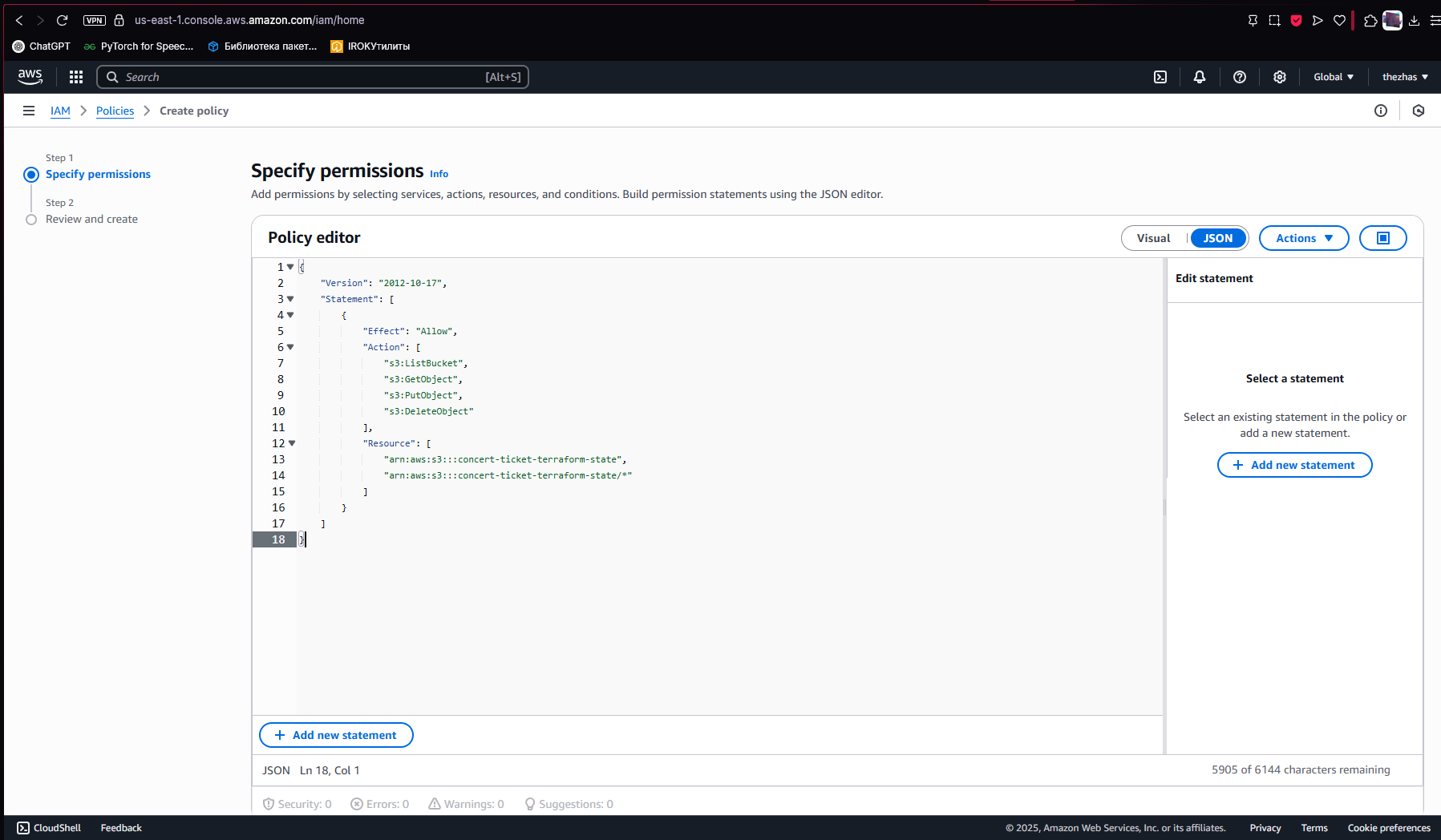
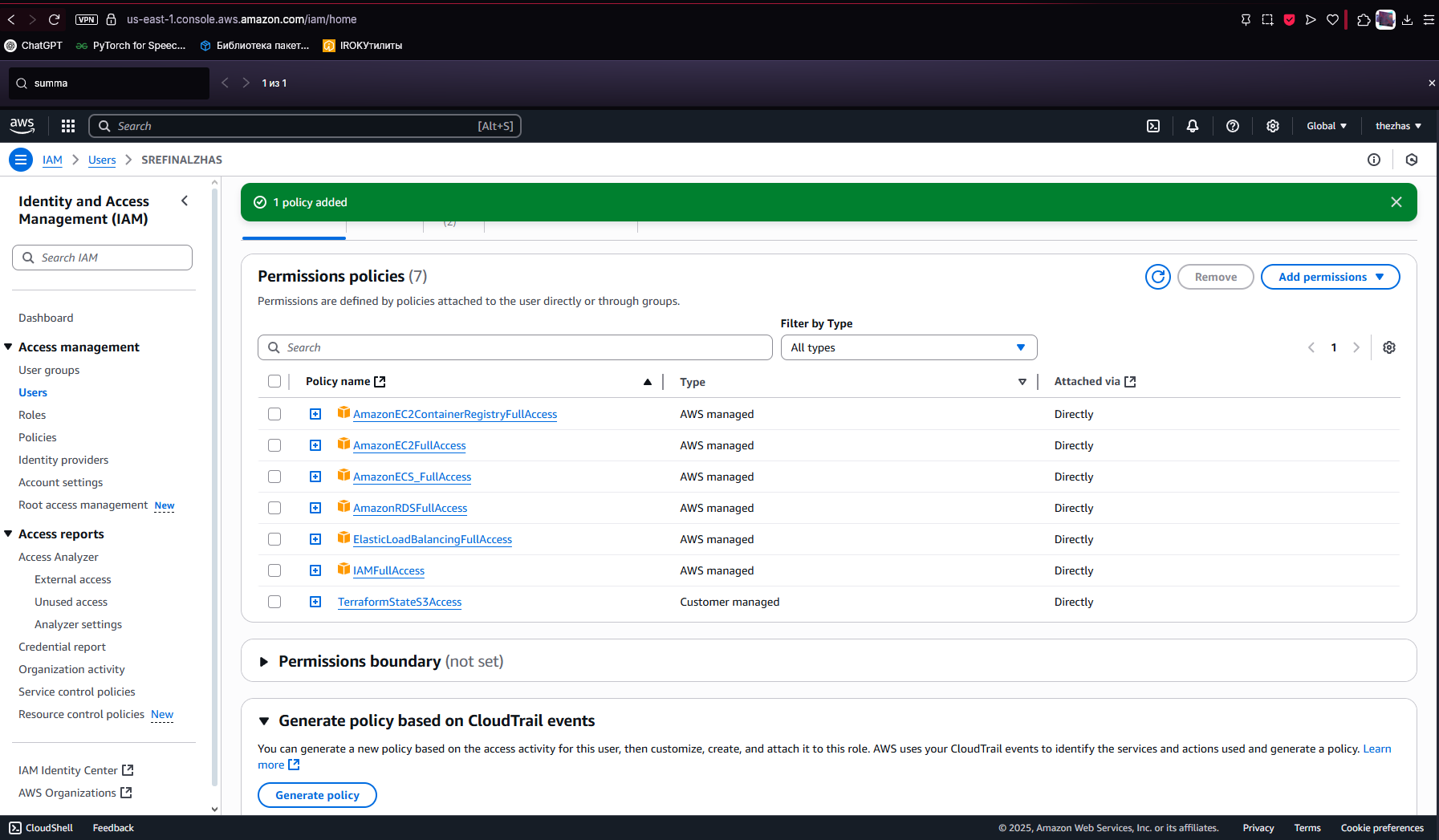
Had big problems with postgre version: Initially tried versions 15.3, 14.7, 13.11. Resolved by using parameter group with PostgreSQL 13. Created custom parameter group instead of default





had very big problems with terraform. Added permissions, even managed my own policy for bucket S3 and added it to my user, only after that I got from error 403 to successful terraform init





Also, Database Configuration resolved admin username being reserved. Changed to "dbadmin" for PostgreSQL compatibility

additions:  
High Availability Design:

* Multiple availability zones used
* Load balanced services
* Database in multi-AZ configuration
* Auto-scaling capabilities built in