

ENPM665 - Classwork Lab 5 - Operation Broken Bridge" Final Report Outline

Name: Raghava Gatadi

UMD ID: raghavag / 121941190

Course & Section: ENPM665, 0101

1. Mission Evidence

```
raghavag@cloudshell:~ (enpm665-demo-project)$ gsutil ls
gs://enpm665-test-bucket-1/
gs://instructions-multi-cloud-gcp-aws-umd/
raghavag@cloudshell:~ (enpm665-demo-project)$ gsutil cp -r gs://instructions-multi-cloud-gcp-aws-umd/ .
Copying gs://instructions-multi-cloud-gcp-aws-umd/mission_briefing.pdf...
Copying gs://instructions-multi-cloud-gcp-aws-umd/multi-cloud-backend.yaml...
Copying gs://instructions-multi-cloud-gcp-aws-umd/source-code.zip...
/[3 files] 3.9 MiB/ 3.9 MiB
Operation completed over 3 objects/3.9 MiB.
raghavag@cloudshell:~ (enpm665-demo-project)$ cd instructions-multi-cloud-gcp-aws-umd/
raghavag@cloudshell:~/instructions-multi-cloud-gcp-aws-umd (enpm665-demo-project)$ ls
mission_briefing.pdf multi-cloud-backend.yaml source-code.zip
raghavag@cloudshell:~/instructions-multi-cloud-gcp-aws-umd (enpm665-demo-project)$ cloudshell download mission_briefing.pdf
raghavag@cloudshell:~/instructions-multi-cloud-gcp-aws-umd (enpm665-demo-project)$
```

```
raghavag@raghava-jumpbox:~$ unzip source-code.zip
Archive: source-code.zip
  creating: static/
  inflating: static/style.css
  creating: templates/
  inflating: templates/index.html
  inflating: templates/mission.html
  inflating: .env.example
  inflating: app.py
  inflating: requirements.txt
raghavag@raghava-jumpbox:~$ ls
app.py requirements.txt source-code.zip static templates
raghavag@raghava-jumpbox:~$ pip3 install -r requirements.txt
```

The screenshot shows the Google Cloud Platform interface for managing VM instances. The top navigation bar includes 'Google Cloud' and a search bar with the query 'insta'. The main content area is titled 'VM instances' and features a 'Create instance' button. Below this, there are tabs for 'Instances', 'Observability', and 'Instance schedules'. The 'Instances' tab is selected. A table lists two VM instances:

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
OK	instance-20251120-013517	us-central1-a			10.128.0.2 (nic0)		SSH
OK	raghava-jumpbox	us-east1-c			10.142.0.2 (nic0)	34.148.112.59 (nic0)	SSH

AWS CloudFront Policies											
Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs	Hit count	Last hit	
allow-5000-from-umd-ip	Ingress	Apply to all	IP ranges:	tcp:5000	Allow	1000	default	Off	-	-	

Aurora and RDS > Databases > multi-cloud-target-db

multi-cloud-target-db

Summary

DB identifier: multi-cloud-target-db	Status: Available	Role: Instance	Engine: PostgreSQL
CPU: 3.91%	Class: db.t3.micro	Current activity: 0 Connections	Region & AZ: us-east-1c

Connectivity & security

Endpoint: multi-cloud-target-db.cmhikak60ejf.us-east-1.rds.amazonaws.com	Networking: Availability Zone: us-east-1c, VPC: vpc-0414686016e6718f0	Security: VPC security groups: Classwork5-BridgeSecurityGroup-P8USbXxsDq0B (sg-0c87805867b8a5b6f), Active: Yes
Port: 5432	Subnet group: default	Publicly accessible: Yes

EC2 > Security Groups > sg-0c87805867b8a5b6f - Classwork5-BridgeSecurityGroup-P8USbXxsDq0B

sg-0c87805867b8a5b6f - Classwork5-BridgeSecurityGroup-P8USbXxsDq0B

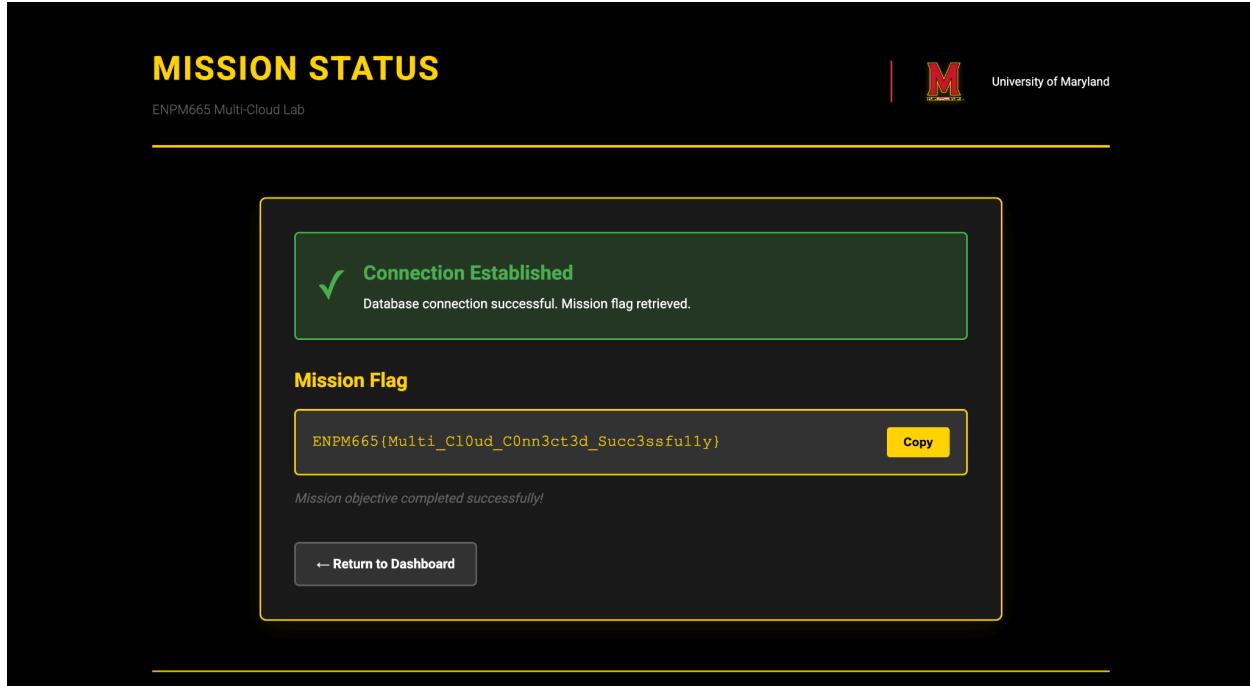
Details

Security group name: Classwork5-BridgeSecurityGroup-P8USbXxsDq0B	Security group ID: sg-0c87805867b8a5b6f	Description: Multi-Cloud Ingress - Pending Configuration	VPC ID: vpc-0414686016e6718f0
Owner: 421113117289	Inbound rules count: 1 Permission entry	Outbound rules count: 1 Permission entry	

Inbound rules

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Des...
-	sgr-057aa0265765abb41	IPv4	PostgreSQL	TCP	5432	34.148.112.59/32	-

```
GNU nano 7.2 .env
DB_HOST=multi-cloud-target-db.cmhikak60ejf.us-east-1.rds.amazonaws.com
DB_USER=postgres
DB_PASS=Raghava7
DB_NAME=postgres
```



2. After Action Report

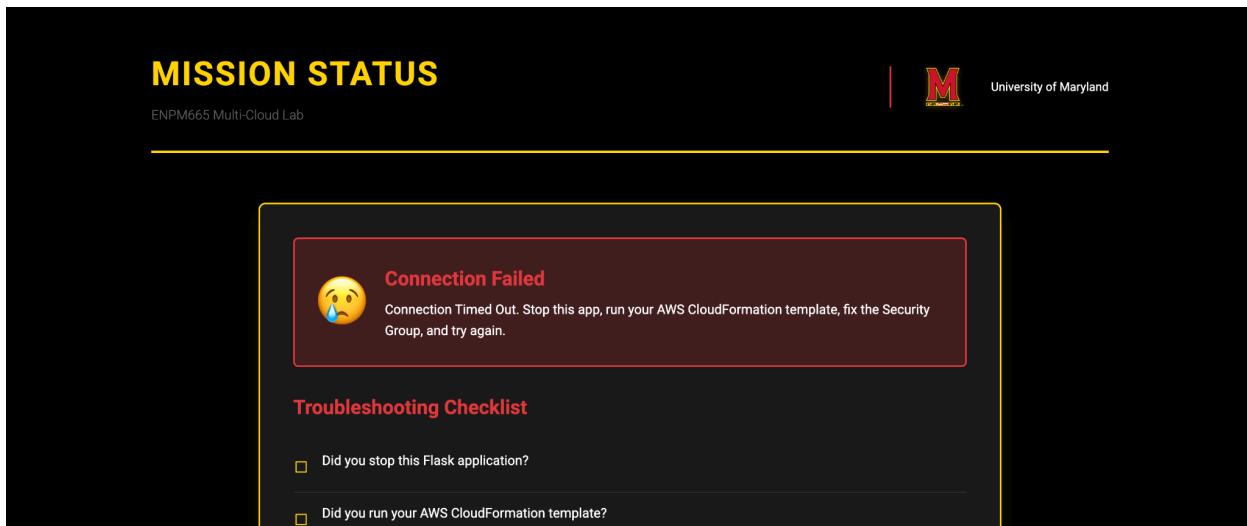
What went well?

Several critical steps of the forensic reconstruction were executed successfully on the first attempt.

1. **Phase I & II Completion:** The initial reconnaissance (finding and exfiltrating artifacts) and the deployment of the AWS CloudFormation backend in the correct us-east-1 region were executed smoothly.
2. **GCP Deployment & Firewall:** The GCP e2-micro VM deployment and the configuration of the GCP Firewall Rule to allow my personal IP (129.2.89.37/32) access to Port 5000 were quickly resolved, which immediately validated that the application was running.
3. **Application Environment Setup:** Successfully using the Python Virtual Environment (venv) to install dependencies without affecting the host OS was a clean deployment practice.

What didn't go well?

The most significant time sink was the mismatch between the common convention (DB_PASSWORD) used in the .env file I created and the application's required variable name (DB_PASS). This caused an authentication failure that presented as a "Connection Timed Out" error, delaying the diagnosis of the actual network issue.



What would you consider doing differently if you had to deploy this again?

1. **Source Code Pre-Review:** Before creating the .env file, I would explicitly examine the application source code (or a dependency library) to confirm the exact environment variable names (DB_HOST, DB_PASS, etc.) to prevent the single-character configuration error that halted the mission.
2. **Strict Networking Focus:** I would make the network link fix—which required adjusting the AWS Security Group's Inbound Rules to allow traffic from the GCP VM's Public IP on port 5432 (the PostgreSQL port)—the first thing I do after the initial AWS deployment. This specific task (the Phase IV "Bridge Engineering" fix) is the critical point in the "Broken Bridge" architecture and should be validated immediately after resource creation.
3. **Automate Deployment:** To minimize human error, I would utilize the VM's Startup Script capability in GCP to automate the entire Phase III deployment: installing unzip, installing Python/VENV, unzipping the code, creating and configuring the .env file,

installing requirements, and running the application. This ensures consistent configuration every time.

Security Posture Assessment

To enhance the security of this multi-cloud environment, a shift away from public-facing resources and static credentials is essential.

Security Domain	Current Posture (Vulnerability)	Proposed Enhancement
Secrets Management	Static, unencrypted password (DB_PASS=Raghava7) stored in a .env file on the VM's disk.	Store the database password in AWS Secrets Manager. The GCP VM's Service Account can be federated with AWS IAM via a role to retrieve a short-lived password directly from Secrets Manager at runtime, removing the credential from the disk entirely.
Network Architecture	GCP VM dashboard (Port 5000) and the AWS RDS Security Group are publicly accessible via whitelisted IPs.	Implement a Zero-Trust Model via Private IPs and IAP. The GCP VM should be created in a private subnet with NO public IP. Access to the dashboard should be via GCP IAP (Identity-Aware Proxy), providing granular, identity-based access control rather than simple IP whitelisting.
Identity (DB Access)	The connection uses a traditional, long-lived username and password (postgres/Raghava7).	Implement IAM Database Authentication. Configure the AWS RDS instance to accept database connections authenticated by an AWS IAM role. The GCP VM would assume this federated role, authenticate with a short-lived IAM token, and eliminate the need for the static DB_PASS entirely.