TECHNICAL DOCUMENTATION

The Jedi Council Web Application

WRITTEN BY ELENA KROUPKIN

JANUARY 2024

TABLE OF CONTENTS

INTRODUCTION

This technical documentation is a comprehensive guide detailing the process of migrating the web applications of the Jedi Council to Amazon Web Services (AWS). The Jedi Council, a local government organization, has been facing significant challenges with their on-premise web applications. This document serves as a roadmap for the migration process, providing detailed instructions and insights to ensure a smooth transition to the cloud.

In addition to this written guide, I have also created a video explanation of the process. Access it here: **link to video**.

STATEMENT OF THE PROBLEM

The Jedi Council maintains two key applications: a public-facing website and an internal tool for recording mission logs. However, these on-premise applications have been plagued by frequent DDoS attacks, scalability issues during planet-wide events, and prolonged downtimes. During times of Republic turmoil, the growing need for the Jedi's wisdom has stressed their current infrastructure to the point of failure. The process of scaling their on-premises infrastructure has proven to be slow and cumbersome, primarily due to the challenges of acquiring additional hardware and the physical setup required to provision new servers.

The goal of this project is to overcome these challenges by migrating the Council's applications to AWS. By leveraging the robust services provided by AWS, the Council aims to manage fluctuations in demand more effectively. This shift promises enhanced scalability, improved security against DDoS attacks, and a more seamless way to manage their growing user base.

FILES USED

A compressed zip file has been prepared, containing the resources of the existing website. This file is available for download and includes the following components:

1. **404.html**: An HTML file displayed when a user tries to access a non-existent page on the website.
2. **index.html**: The main HTML file that serves as the home page of the website.
3. **Images and Logos**: Six visual elements used throughout the website.

The zip file can be downloaded from the provided link and will be used in the process of migrating the website to AWS.

https://storage.googleapis.com/nextwork_course_resources/courses/aws/aws_jedi_council_static_www.zip

S3 BUCKET

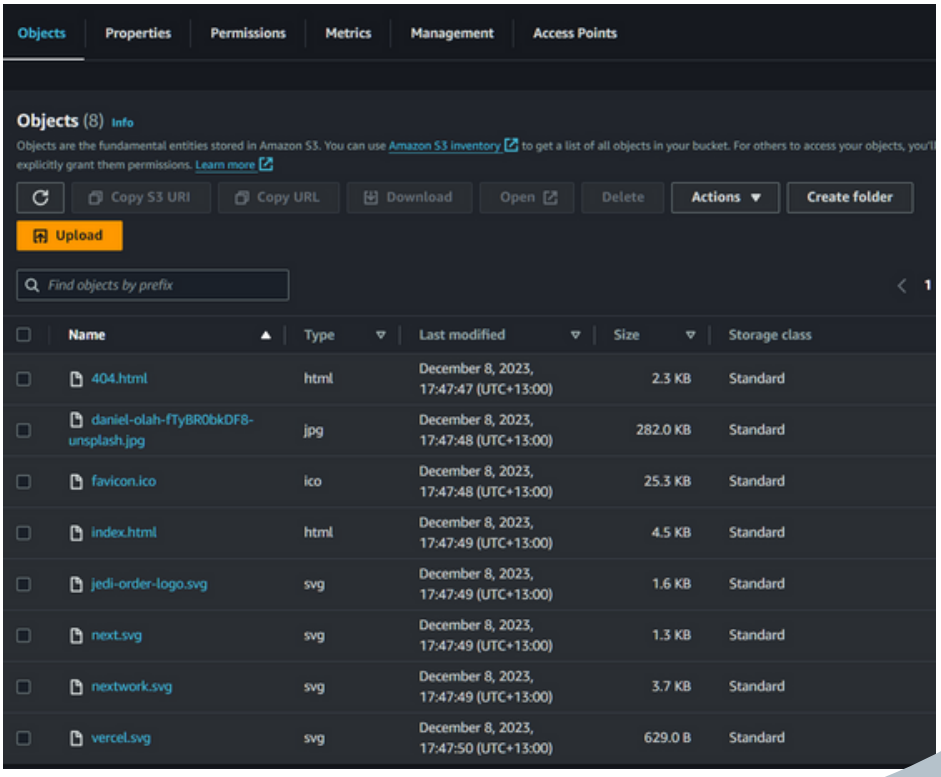
An S3 bucket is required to store a static website files, as it offers a secure, scalable, and reliable storage solution that can be accessed globally via the web.

Bucket Creation

1. Access the Amazon S3 service within the AWS Management Console.
2. Initiate bucket creation by clicking the "Create bucket" button.
3. Assign the bucket a name: "**council-of-the-jedi-www**".
4. Select a region that aligns with your target audience for optimal performance.
5. Leave the remaining settings as default unless specific customization is required.
6. Click "Create".

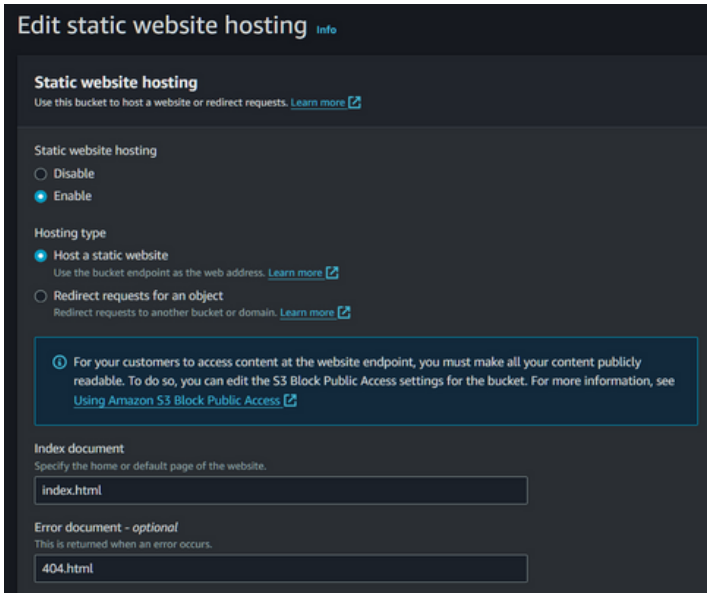
Uploading Website Files

1. Extract the website's content from the provided zip file.
2. Within the newly created S3 bucket, locate and utilize the "Upload" button to transfer the extracted files.
3. Ensure successful completion of the upload process.



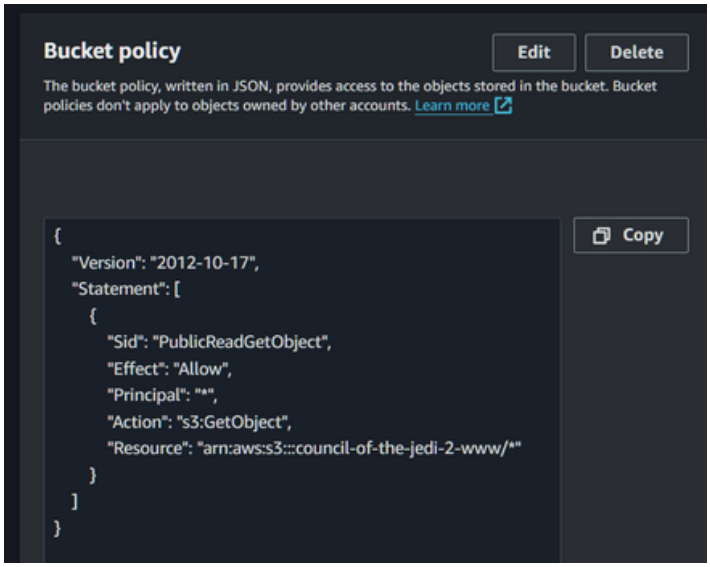
Enabling Static Website Hosting

1. Select the bucket and proceed to its "Properties" tab.
2. Navigate to the "Static website hosting" section and enable it by switching its status from "Disabled" to "Enabled".
3. Configure the "Index document" field to point to "index.html".
4. Similarly, designate "404.html" as the "Error document" for handling potential errors.



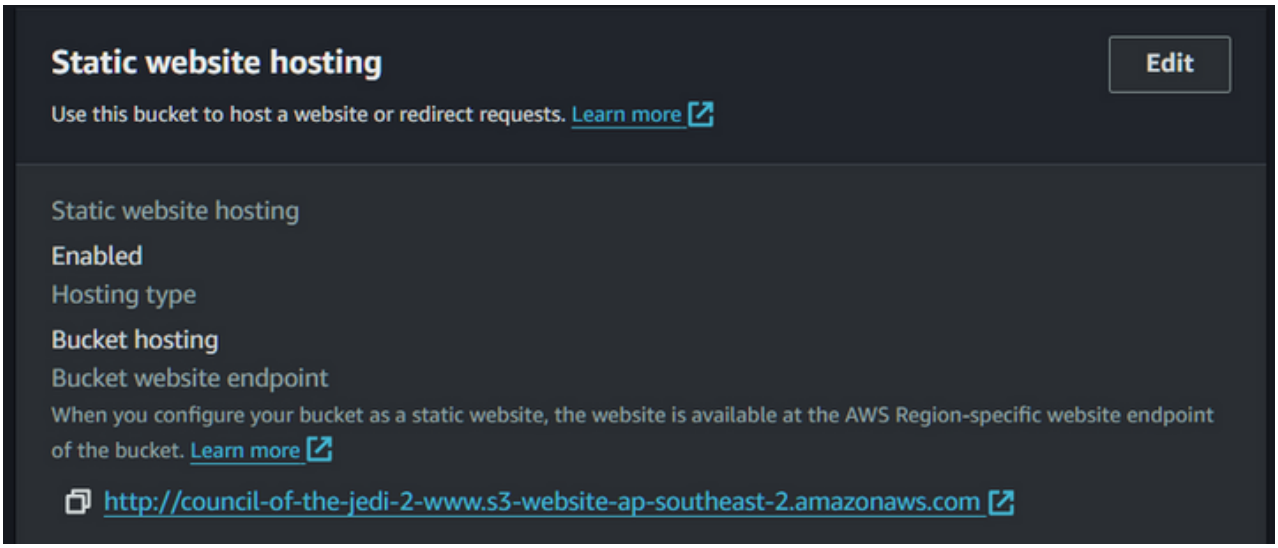
Configuring Bucket Permissions

1. Access the "Permissions" tab within the bucket's settings.
2. Set the bucket policy to "**PublicReadGetObject**" to grant public access to the website's content.
3. Save the policy to activate the accessibility settings.

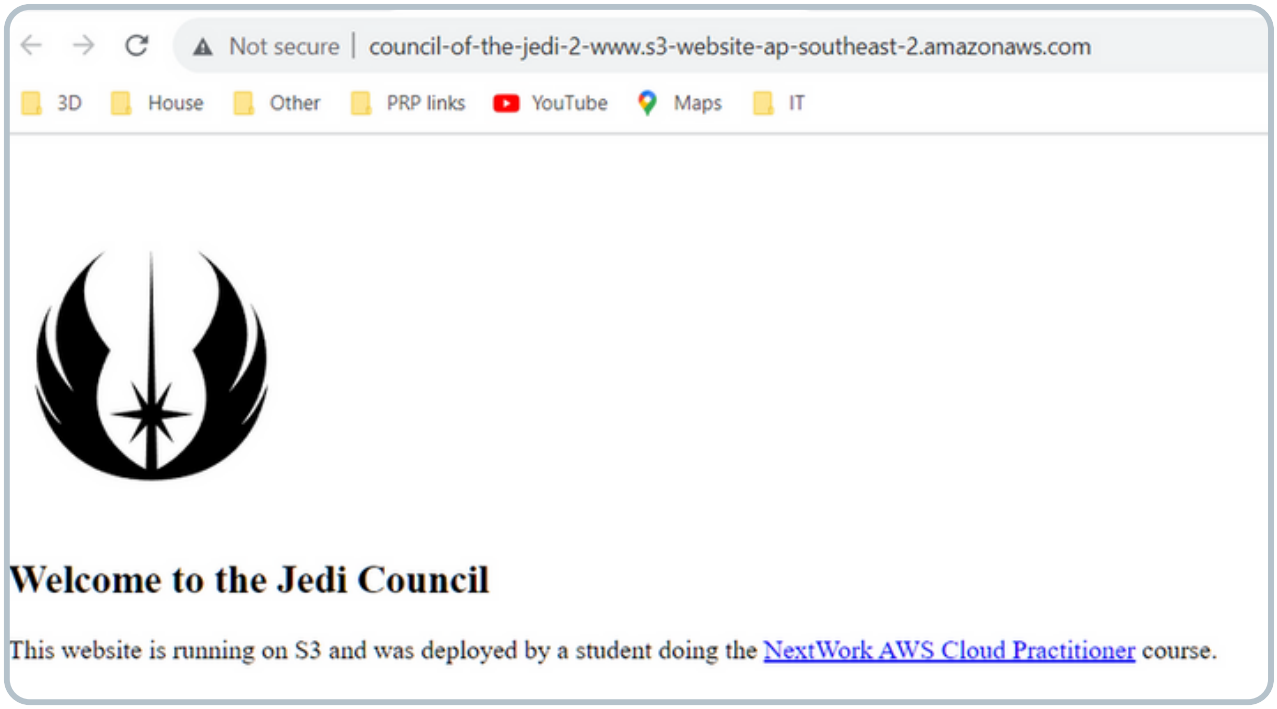


Verify the Static Website is running

1. Select the Buckets and proceed to its "Properties" tab.
2. Navigate to the "Static website hosting" section and click on the Bucket website endpoint



the Static Website is running

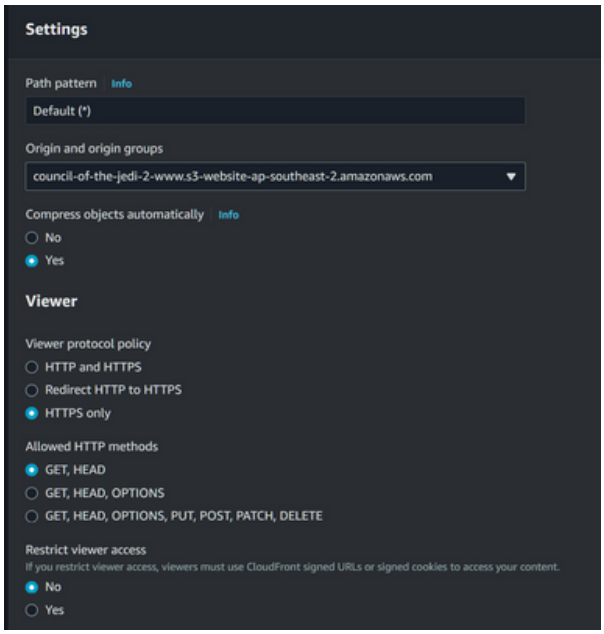


Establishing a CloudFront Distribution

A CloudFront distribution is needed for fast, secure global delivery of a static website content, as it leverages a global network of edge locations to ensure low latency and high data transfer speeds.

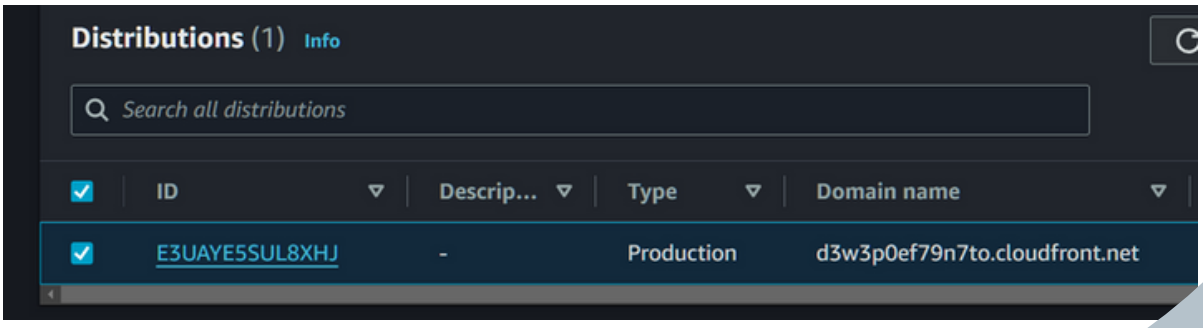
Distribution Configuration

1. Access the CloudFront service within the AWS Management Console.
2. Initiate the creation of a new distribution.
3. Specify the S3 bucket created in Step 1 as the origin. Configure:
   - Restrict incoming traffic to HTTPS only.
   - Block all HTTP methods except GET and HEAD.
4. [Optional] For enhanced website protection, enable the WAF. Be mindful of associated costs.



Domain Routing

The domain name automatically assigned to the S3 bucket for public access after enabling static website hosting.



VPC Creation

The VPC is required to create a secure, isolated, and customizable virtual network for AWS resources.

VPC Creation

1. Access the VPC service within the AWS Management Console.
2. Initiate the creation of a new VPC.
3. Assign a descriptive name to the VPC, such as "**jegi5-vpc**".
4. Select an appropriate IPv4 CIDR block for the VPC, considering future scalability needs **10.0.0.0/16**.
5. Temporarily leave other settings as default.

Security Group Configuration

1. Navigate to the Security Groups section within the VPC service.
2. Create a new security group tailored for the **Mission Logs** application.
3. Define inbound rules as follows: Authorize all inbound HTTP traffic (port 80) from any source.
4. Allow all outbound traffic to enable responses.

Security Groups

	name	type	protocol	port range	source/ destination
Public HTTP allows all access	sg1-public-http-public	IN: HTTP OUT: All traffic	IN: TCP OUT: All	IN: 80 OUT: All	IN: 0.0.0.0/0 OUT: 0.0.0.0/0
Public SSH allows all access	sg2-public-ssh-public	IN: SSH OUT: All traffic	IN: TCP OUT: All	IN: 22 OUT: All	IN: 0.0.0.0/0 OUT: 0.0.0.0/0
Private RDS-EC2 SG attached to rds-jedi-mission-logs to allow EC2 instances with specific security groups attached to connect to the database	rds-ec2-1	IN: PostgreSQL OUT: ----	IN: TCP OUT: ----	IN: 5432 OUT: ---	IN: sg-0e3b13... / ec2-rds-1 OUT: ----
Private EC2-RDS SG attached to instances to securely connect to db-jedi-mission-logs	ec2-rds-1	IN: ---------- OUT: PostgreSQL	IN: ---------- OUT: TCP	IN: ------ OUT: 5432	IN: ---------- OUT: sg-062d... / rds-ec2-1

Public Subnet Creation

1. Return to the VPC dashboard and create two public subnets within the newly created VPC.
2. Distribute the subnets across distinct Availability Zones for enhanced availability and redundancy.

Name	Subnet ID	State	VPC	IPv4 CIDR
RDS-Pvt-subnet-1	subnet-0c5e2871b8b3f51fe	Available	vpc-0be710af13edbfc52 | jedi5-vpc	10.0.0.128/25
RDS-Pvt-subnet-2	subnet-0ad03eddf2c7ff91c	Available	vpc-0be710af13edbfc52 | jedi5-vpc	10.0.3.0/25
sub2-b-private	subnet-0f06595ef902bc303	Available	vpc-0be710af13edbfc52 | jedi5-vpc	10.0.2.0/24
sub1-a-public	subnet-094ff3d0568248efa	Available	vpc-0be710af13edbfc52 | jedi5-vpc	10.0.1.0/24
sub3-c-private	subnet-07f47290bbf592acf	Available	vpc-0be710af13edbfc52 | jedi5-vpc	10.0.4.0/24
RDS-Pvt-subnet-3	subnet-007717637389d6a3b	Available	vpc-0be710af13edbfc52 | jedi5-vpc	10.0.3.128/25

Internet Gateway Setup

1. Create an internet gateway within the VPC service.
2. Attach the internet gateway to the **jedi5-vpc**.

Details Info

Internet gateway ID	State	VPC ID	Owner
igw-0fe62de7f9f01fde8	Attached	vpc-0be710af13edbfc52 | jedi5-vpc	909597544674

Tags Manage tags

Search tags

Key	Value
Name	jedi-gw

Public Route Table Configuration

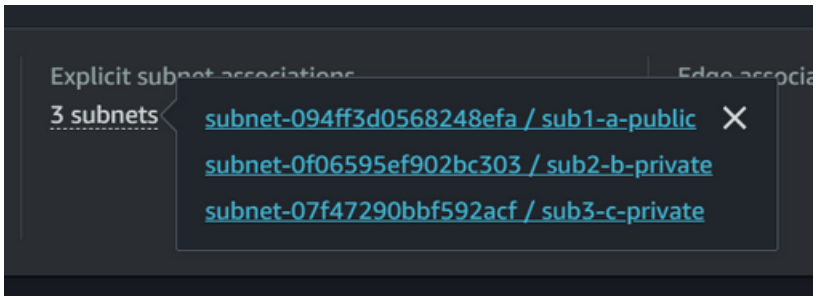
1. Create a public route table for each public subnet.
2. Within each route table, establish a route directing all traffic (0.0.0.0/0) to the internet gateway, enabling internet access.
3. Associate each public subnet with its corresponding public route table.

Public Route Table

Routes (2)

Filter routes

Destination	Target	Status	Propagated
0.0.0.0/0	igw-0fe62de7f9f01fde8	Active	No
10.0.0.0/16	local	Active	No

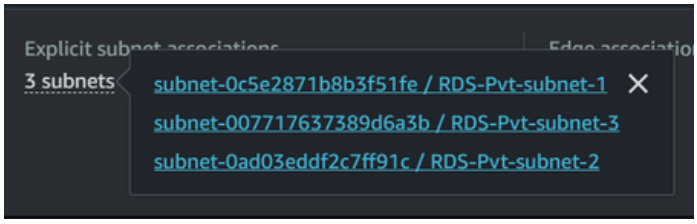


Private Route Table

Routes (1)

Filter routes

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No



Launching an EC2 Instance

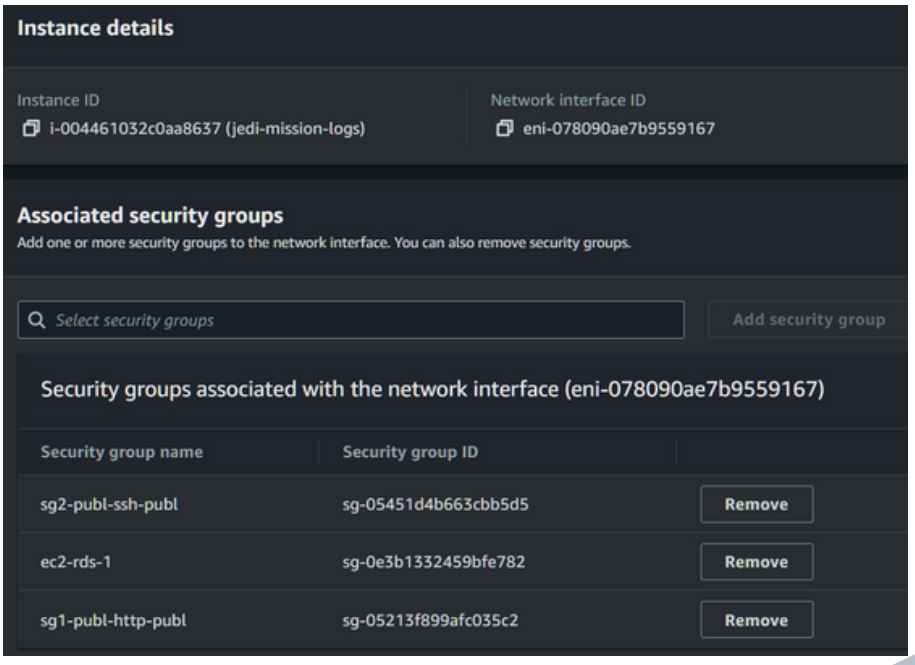
EC2 is the foundation for building and running virtually any application in the cloud. It provides a quick and efficient creation and manage a dynamic, scalable, and cost-effective computing infrastructure in the cloud.

Instance Initiation

1. Access the EC2 service within the AWS Management Console.
2. Launch a new EC2 instance.

Instance Configuration

1. Assign a descriptive and memorable name to the instance, such as "**jedi-mission-logs**".
2. Select the Amazon Linux AMI as the operating system.
3. Choose the **t2.micro** instance type to align with the free tier.
4. Configure the instance to reside within the previously created **jedi5-vpc**.
5. Select one of the public subnets within the VPC for instance placement.
6. Assign the security group tailored for the Mission Logs application to the instance.



Network Access

1. Under the "Configure Security Group" section, ensure the following inbound rule is present:
2. HTTP traffic (**port 80**) from any source (**0.0.0.0/0**)
3. Verify that all outbound traffic is permitted for proper response handling.

Inbound rules

Filter rules

Name	Security group rule ID	Port range	Protocol
–	sgr-02efa6205b1a9efd4	80	TCP
–	sgr-06255bd1bc81b302e	22	TCP

Outbound rules

Filter rules

Name	Security group rule ID	Port range	Protocol
–	2 IDs	All	All
–	sgr-054dfa2aa39b59d24	5432	TCP

Instance Launch

1. Review the configuration settings and initiate the instance launch process.
2. Once initiated, monitor the instance's status until it transitions to a "**running**" state.

Name	Instance ID	Instance state	Instance type
jedi-mission-logs	i-004461032c0aa8637	Running	t2.micro

Provisioning a PostgreSQL Database with RDS

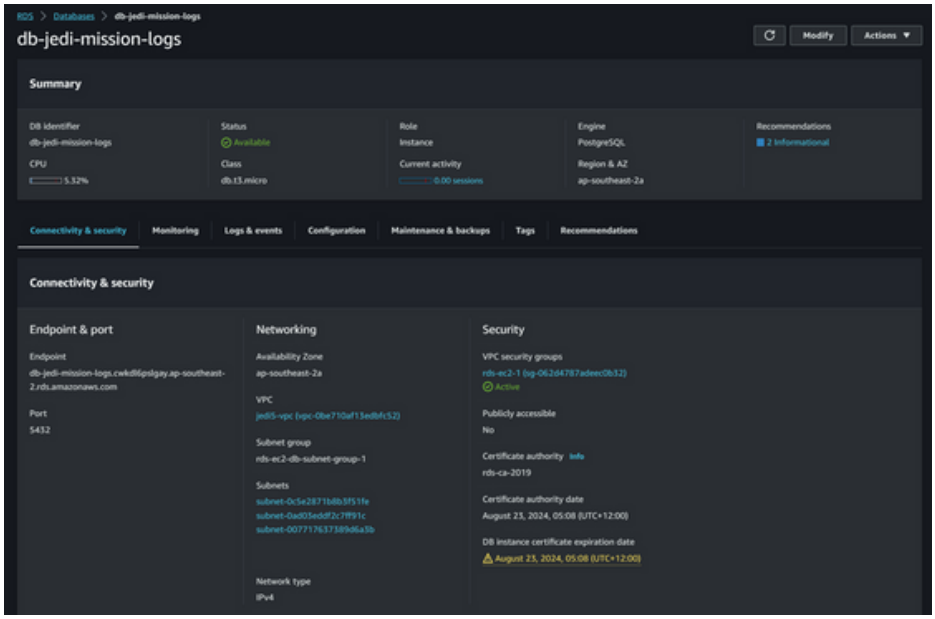
Provisioning a PostgreSQL database with RDS gives you a managed, scalable, and secure database solution in the cloud, freeing you from infrastructure headaches and focusing on your application.

Database Creation

1. Access the RDS service within the AWS Management Console.
2. Initiate the creation of a new database.
3. Select **PostgreSQL** as the database engine.
4. Choose the "Free Tier" template to adhere to cost constraints.
5. Assign a descriptive identifier to the database, such as "**db-jedi-mission-logs**".

VPC and Security Group Configuration

1. Configure the database to reside within the **jedi5-vpc**.
2. Select one of the **public** subnets within the VPC for database placement.
3. Create a new VPC security group specifically for the database connection.
4. Within the security group, define inbound rules to permit access from the **Mission Logs** application's EC2 instance.

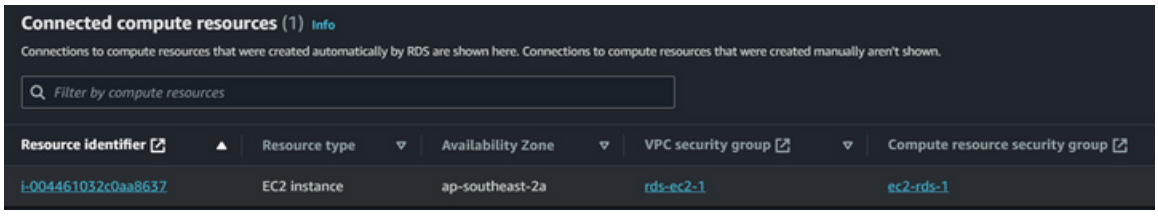


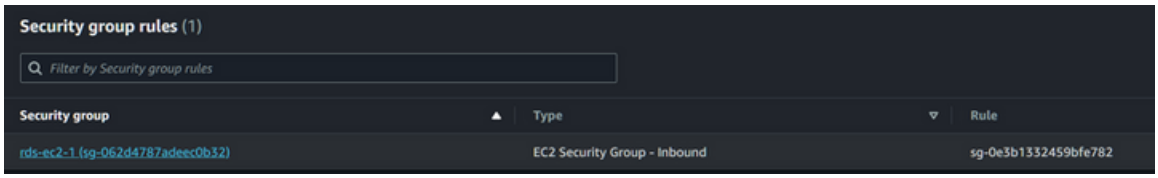
Database Credentials

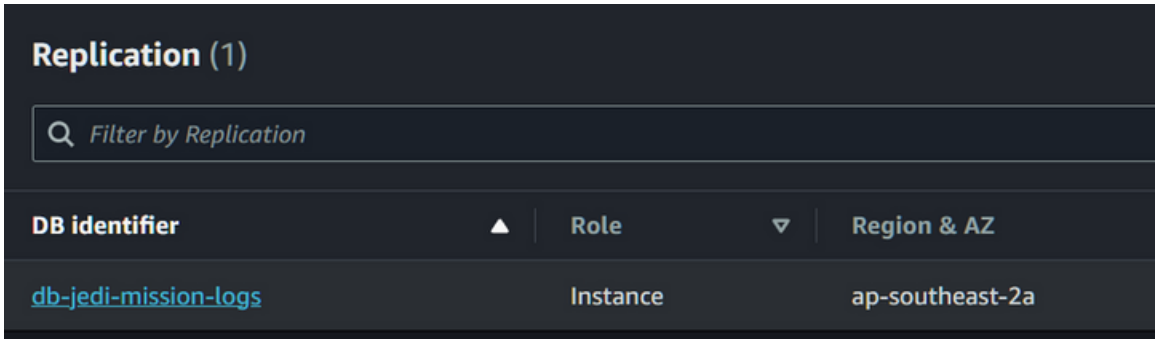
1. Set a default password for the RDS instance.
2. Actions: setup EC2 connection - select existing EC2
3. Give DB name
4. Note the generated instance URL for the PostgreSQL database.

Database Connection

1. Upon database creation, establish a connection between the EC2 instance and the RDS database.
2. Provide the database endpoint, username, and password during the connection process.





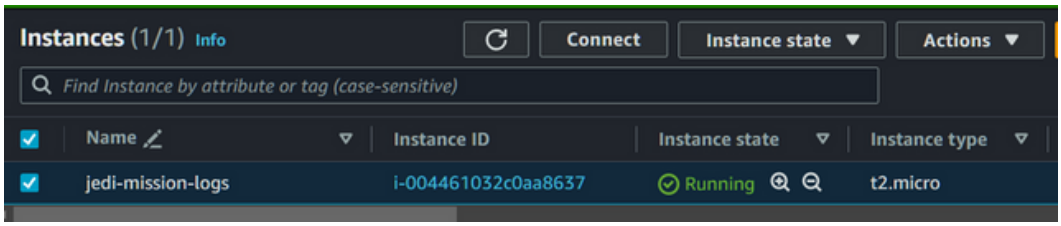


Installing Docker and Enabling the Application

Docker streamlines deployment, isolates services, and enhances portability, making an app containerized, secure, and flexible.

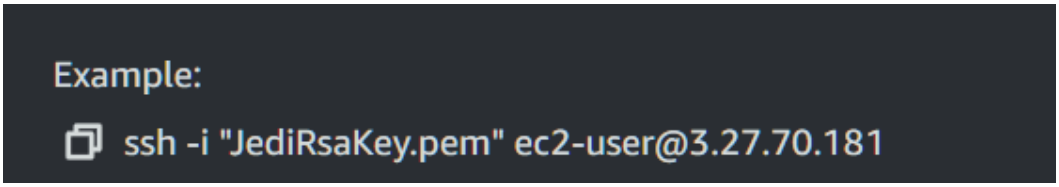
Instance Connection

1. From the EC2 dashboard, right-click on the instance launched in previous step.
2. Select "Connect" to establish an SSH session.

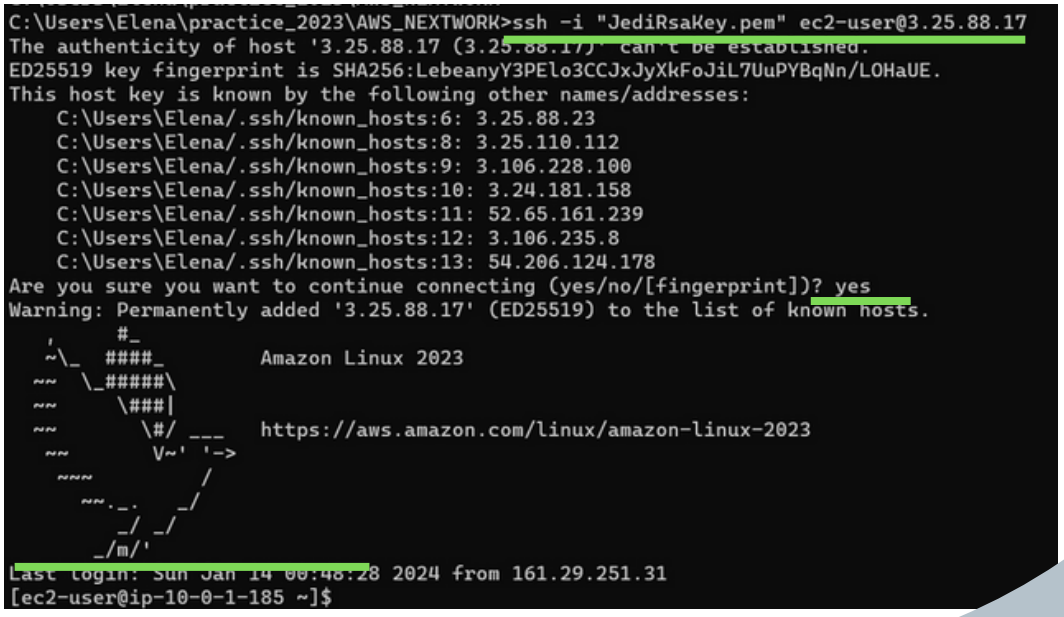


Docker Installation

1. In Command Prompt : cd to folder with Key Pairs
2. Select SSH client tab to copy the connection link



- Past the path from SSH. Type “**yes**” .
- The bird image means - connected



- Within the Command Prompt, execute the following commands sequentially to install and configure Docker:

```
sudo yum update
sudo yum install docker
sudo usermod -a -G docker ec2-user
sudo systemctl enable docker.service
```

- Verify successful Docker installation by running:

```
docker --version
```

Launching the Jedi Mission Logs Application in Docker

Image Retrieval

1. Retrieve the pre-built Docker image for the Jedi Mission Logs application from the ECR repository:

```
docker pull
public.ecr.aws/z7j4c9h0/nextwork/course
work/real-world-projects/jedi-mission-
logs:latest
```

Container Deployment

1. Launch the application in a Docker container, establishing database connectivity and port mapping

DB password DB endpoint DB name

```
docker run -d -p 80:3000 -e DATABASE_URL=postgres://postgres:
<your password here>@<DB instance URL here>:5432/jedi-mission-
logs public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-
projects/jedi-mission-logs:latest
```

Verification

1. If successful, the application should be accessible via the **EC2 instance's public IP address**.
2. The public website should be reachable through the **S3 URL** established in previous steps.
3. In Command Prompt:

```
docker image ls
docker ps
```

should print list of dockers and images

Create a Launch Template and AMI from your EC2 Instance

1. Create an Image from your EC2 instance.
2. Create a Launch Template from your EC2 instance.
3. Give a name
4. select Amazon Linux
5. Enable creating autoscaling groups .
6. Select t2.micro type
7. Add RSA Key Pair
8. Add public subnet
9. Add security groups: public-HTTP, public-SSH, private-EC2-RDS
10. Add the commands to the “User Data” field in Advaned details.

User data - *optional* Info

Upload a file with your user data or enter it in the field.

Choose file

```
#!/bin/bash
docker pull public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-projects/jedi-mission-logs:latest
docker run -d -p 80:3000 -e DATABASE_URL=postgres://postgres:<your password here>@<DB instance URL here>:5432/jedi-mission-logs public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-projects/jedi-mission-logs:latest
```

Security rules

Inbound rules

Filter rules

Name	Security group rule ID	Port range	Protocol	Source
–	sgr-02efa6205b1a9efd4	80	TCP	0.0.0.0/0
–	sgr-06255bd1bc81b302e	22	TCP	0.0.0.0/0

Outbound rules

Filter rules

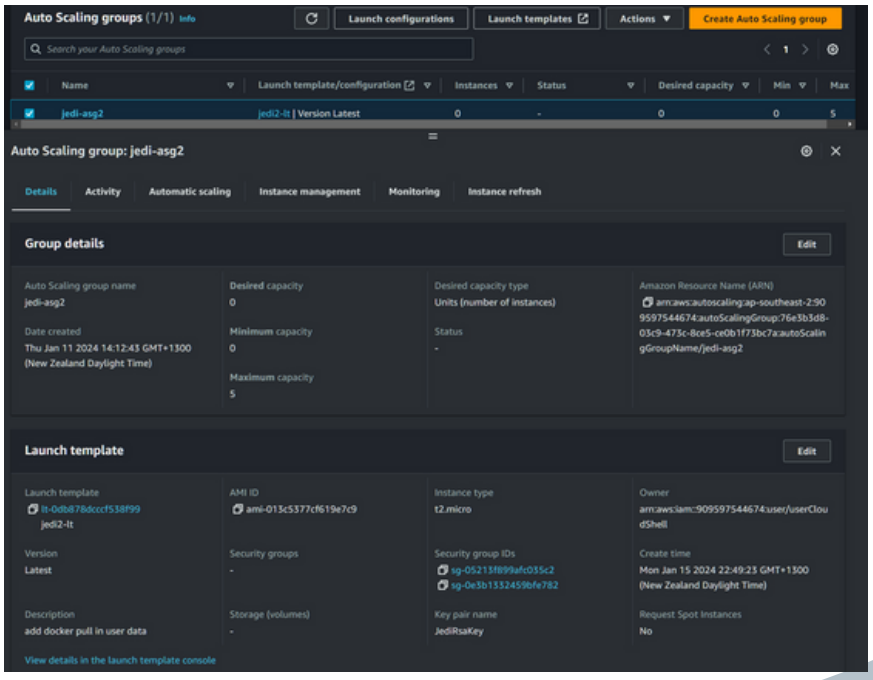
Name	Security group rule ID	Port range	Protocol	Destination
–	2 IDs	All	All	0.0.0.0/0
–	sgr-054dfa2aa39b59d24	5432	TCP	sg-062d4787adeec0b32

Create the Load Balancer

1. From EC2 Dashboard select Load Balancers, Create load balancer
2. Select Application Load Balancer
3. select Internet-facing Scheme
4. In Mappings : select 3 Availability Zones
5. Select public security group (publ-http-publ)
6. In Listeners and routing Create target group (in different tab)
   - instances
   - give a name
7. Back to Load balancer, refresh Listeners and routing Default action, Select the new target group

Create the Auto Scaling Group

1. From EC2 Dashboard select Auto Scaling Groups : Create an Auto Scaling group
2. Add Launch Template .
3. Subnet public, zone a
4. Attach to an existing load balancer
5. Turn on Elastic Load Balancing health checks
6. Enable group metrics collection within CloudWatch
7. Group size: Desired capacity 1, Min 0, Max 5.

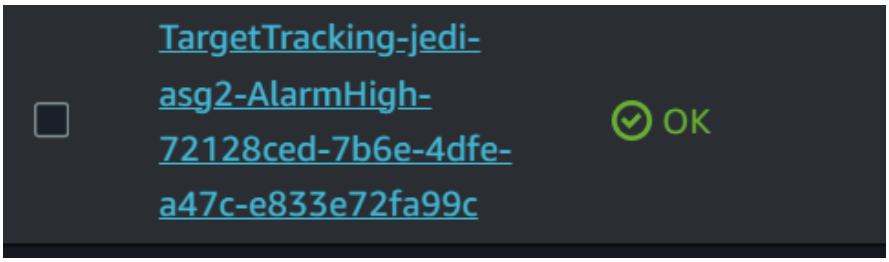


CHECK YOUR WORK

Auto Scaling Test

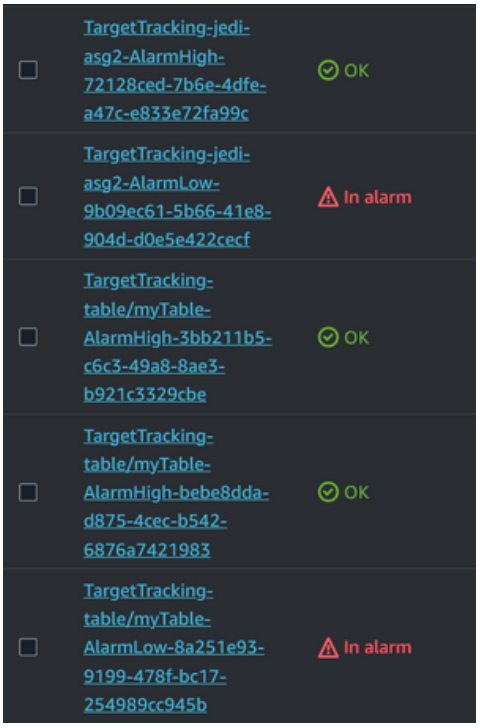
Verify Initial State:

1. Access the AWS Management Console.
2. Navigate to the CloudWatch console.
3. Under Alarms, choose All alarms.
4. Locate the alarm named **AlarmHigh** and ensure its state is OK. This indicates CPU utilization is below the threshold.



Monitor Alarm Response:

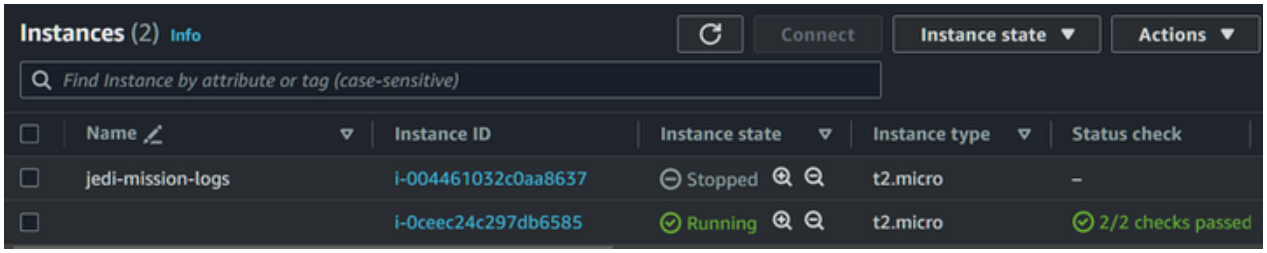
1. Return to the CloudWatch Management Console.
2. Wait for approximately 3 minutes to observe the effect of the load.
3. Refresh the console to view the AlarmHigh chart.
4. The chart should depict rising CPU utilization.
5. Within roughly 5 minutes, the AlarmHigh alarm status should change to In alarm, signaling auto-scaling initiation.



Auto Scaling Test

Confirm Auto-Scaling Action:

1. Access the EC2 console.
2. Under Instances, choose Instances.
3. Refresh the page.
4. More than two instances of **jedi-mission-logs** Instance should be visible, demonstrating that auto-scaling has successfully added instances in response to the alarm.
5. Refresh for updates: Regularly refresh the CloudWatch and EC2 consoles to visualize the auto-scaling process.



Post-Deployment Health Checks

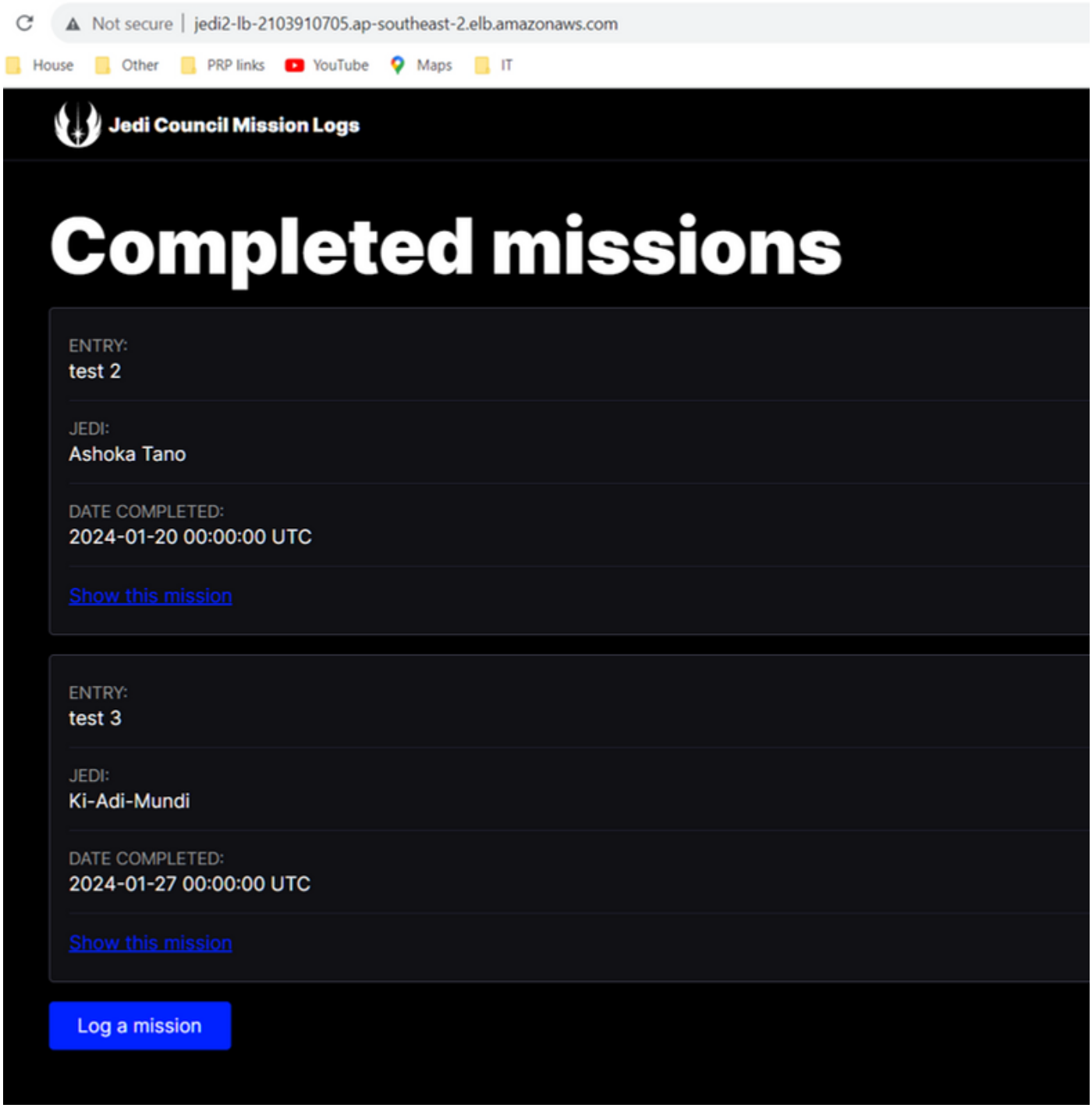
To verify the health of the new instance:

1. **Confirm Docker Pull and Run:**
   - Within the instance's User Data, ensure the presence of code for both Docker pull and Docker run operations.
2. **Monitor Target Group Health:**
   - Access the Target Group associated with the instance.
   - Observe the instance's health status.
   - Allow sufficient time for it to transition to a healthy state.
3. **Access the Application:**
   - Navigate to the Load Balancer section.
   - Under Details, locate and copy the DNS name.
   - Open a new browser window.
   - Paste the DNS name into the address bar.
   - Press Enter to access the working application.

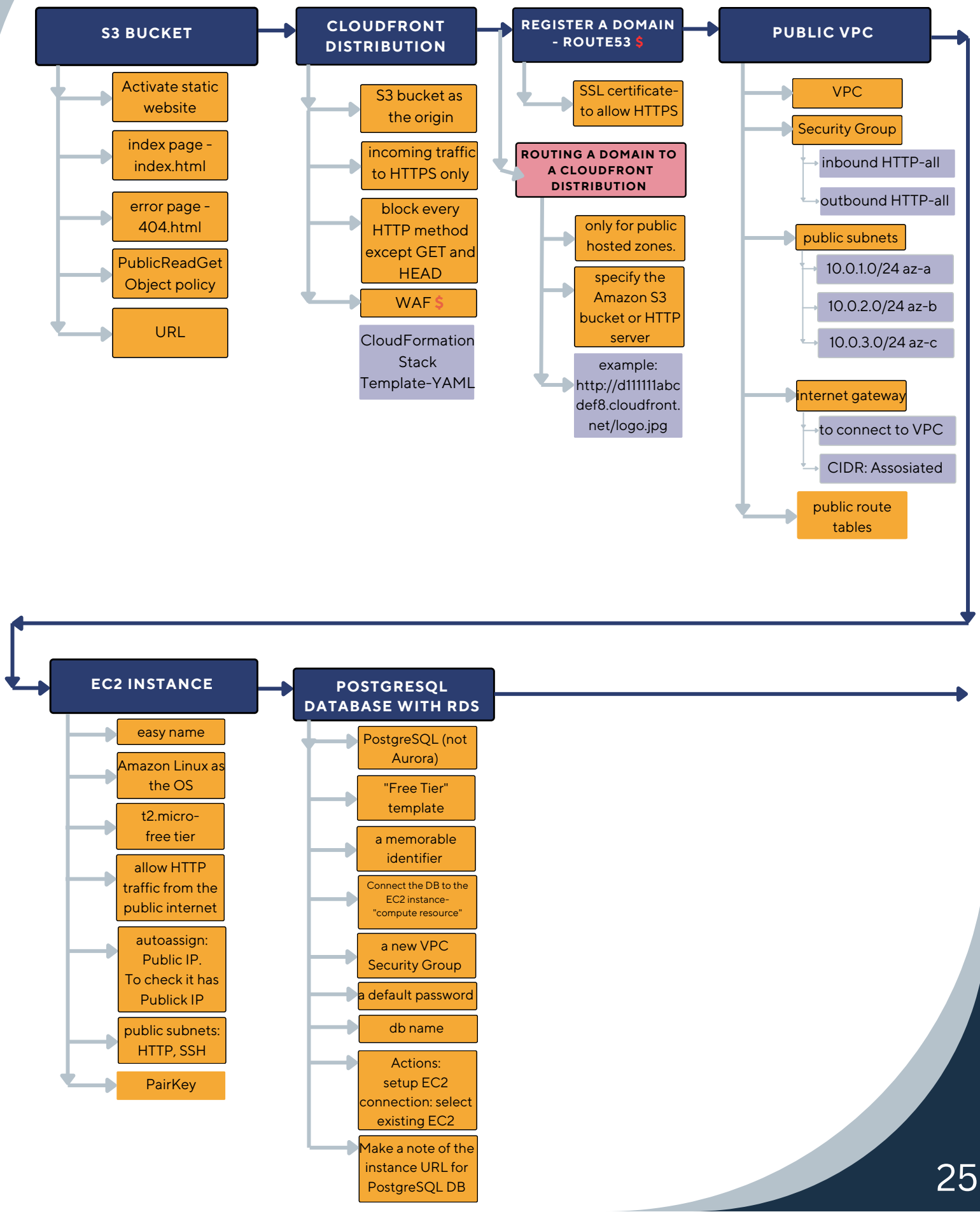
Additional Considerations:

- Time for Health Checks: Acknowledge that health check processes might require a few minutes to complete.
- Troubleshooting: If health checks fail or the application remains inaccessible, initiate troubleshooting procedures.
- Logging and Monitoring: Employ logging and monitoring tools to track instance health and application performance.

Post-Migration Verification:
Visual Confirmation of Successful Transition



Visualizing the Jedi Cloud Journey: A Flowchart



Visualizing the Jedi Cloud Journey: A Flowchart

INSTALL DOCKER

from EC2 instance / Connect/SSH copy connection path

Example:
ssh -i "JediRsaKey.pem" ec2-user@52.65.181.157

from Command Prompt
- cd to folder with Pair Keys
- past the path from SSH
- "yes"
- bird = connec

installl docker

```
sudo yum update
sudo yum install docker
sudo usermod -a -G docker ec2-user
sudo systemctl enable docker.service
```

Verify docker is running

```
docker --version
```

copy the application docker IMG to Ec2 instance

copy IMG

```
#!/bin/bash
docker pull public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-projects/jedi-mission-logs:latest
```

This tells Docker to download the image for the Jedi Mission Logs website from the ECR repository.

Verify docker IMG is there

```
docker image ls
```

Run (DB password DB endpoint DB name

run container from the IMG

```
docker run -d -p 80:3000 -e DATABASE_URL=postgres://postgres:<your password here>@<DB instance URL here>:5432/jedi-mission-logs
public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-projects/jedi-mission-logs:latest
```

Verify container with docker IMG is running

```
docker ps
```

check if if works

Open in a browser IP public from EC2 (http://54.206.124.178/)

LAUNCH TEMPLATE AND AMI

- Image from EC2 instance
- Launch Template from EC2 instance
  - for autoscaling groups
  - User Data

```
#!/bin/bash
docker pull public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-projects/jedi-mission-logs:latest
docker run -d -p 80:3000 -e DATABASE_URL=postgres://postgres:<your password here>@<DB instance URL here>:5432/jedi-mission-logs public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-projects/jedi-mission-logs:latest
```

DB password DB endpoint DB name

Visualizing the Jedi Cloud Journey: A Flowchart