
Jedi Council Migration

Last Updated 20/12/2023

Written by Z'Arn Payne

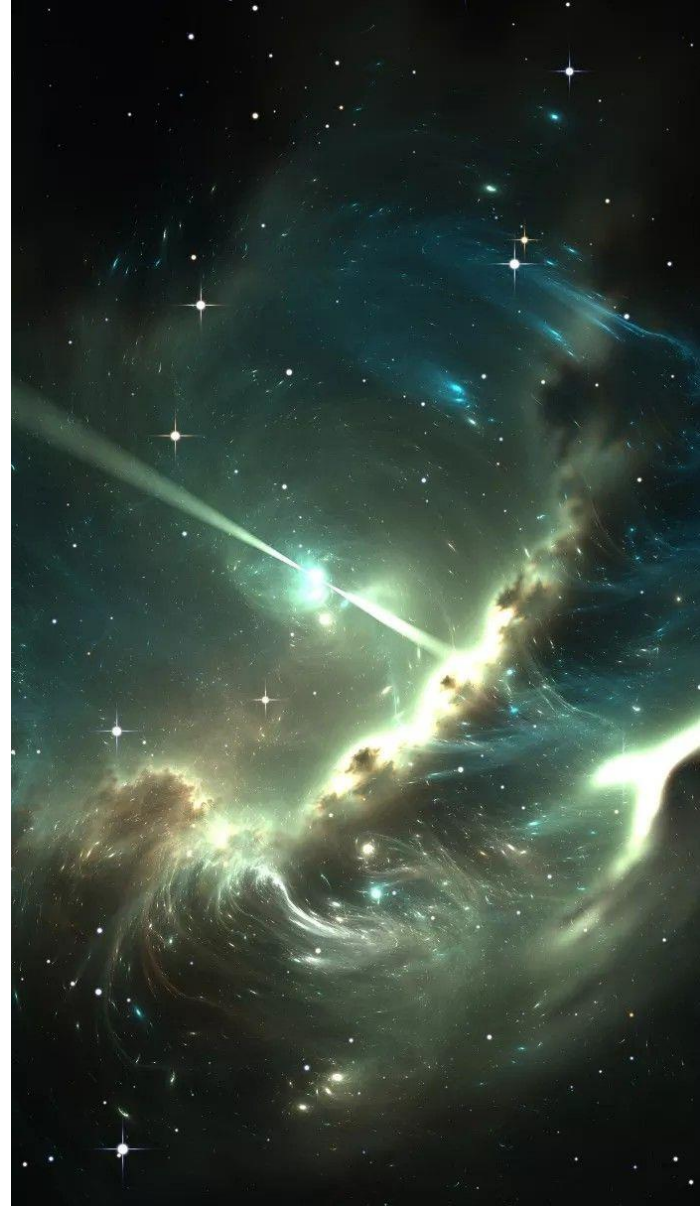


Table of Contents

Table of Contents.....	2
Background	3
Executive Summary	3
Drivers for Change	3
Design – Website.....	3
Design – Mission Log	4
Existing Files.....	4
Migrating to S3	5
Creation of S3 Bucket	5
General Configuration	5
Permissions and Policy	5
Management of CloudFront Distribution.....	6
Creating Certificate.....	6
Route53 Setup.....	6
AutoScaling Logging System	8
Creation of our Virtual Private Cloud (VPC)	8
Subnetting	8
Route Tables and Internet Gateways	8
Creating Instances	9
Configuring Security Groups	9
Creating our First instance	9
Creation of Relational Database Services (RDS)	10
Deploying Service and Creating AutoScaling Groups.....	10
Deploying the Service.....	10
Creation of Image and Launch Template	10
Creating the Load Balancer and Target Groups.....	11

Background

Executive Summary

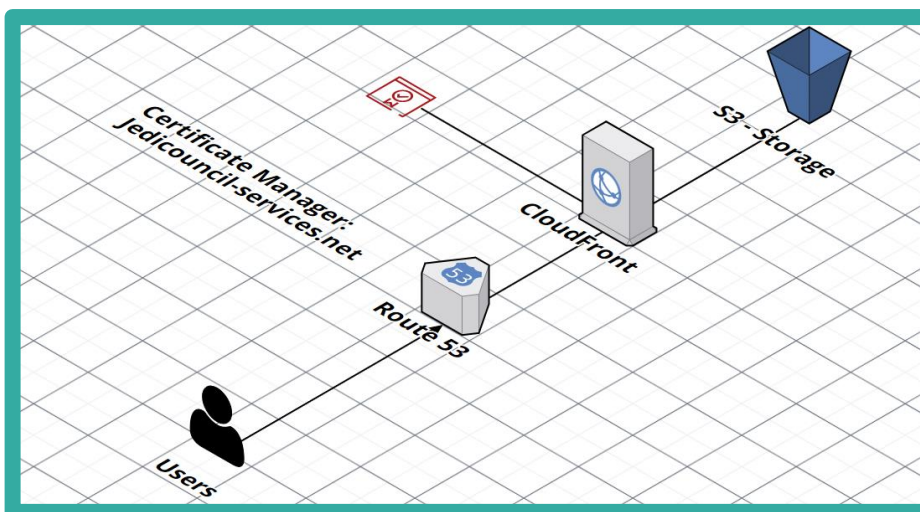
This document contains all the steps undergone to migrate the Jedi Council's webservices to the AWS platform. This has been done in two major steps: migration of the website to an S3 service. The other step is to deploy the mission log through docker into an Autoscaling EC2 instance. For a demonstration of the completed service, please see the following [video](#).

Drivers for Change

By offering IaaS solutions to the problems faced by the Jedi-Council, they can achieve a more decentralised design for their services. This allows them to scale their services based on their current needs. They also no longer must manage their infrastructure on-site, allowing them greater flexibility.

They have also been suffering from extremely frequent DDos attacks. Amazon provides many services to help mitigate these issues for its customers.

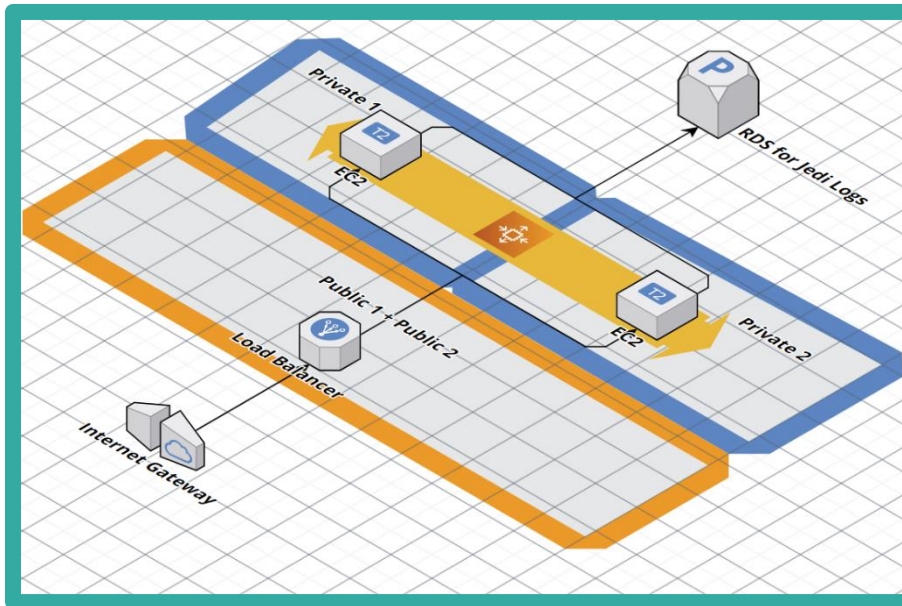
Design – Website



The following is the design for the website. Here we manage access to the S3 bucket through a CloudFront distribution.

The distribution has DNS traffic managed by Route53.

Design – Mission Log



Here is the Design for the Autoscaling mission log system.

Here, it uses an RDS to store and share information to scalable EC2 services. Note that the load balancer is the only form of access to the instances and hence database.

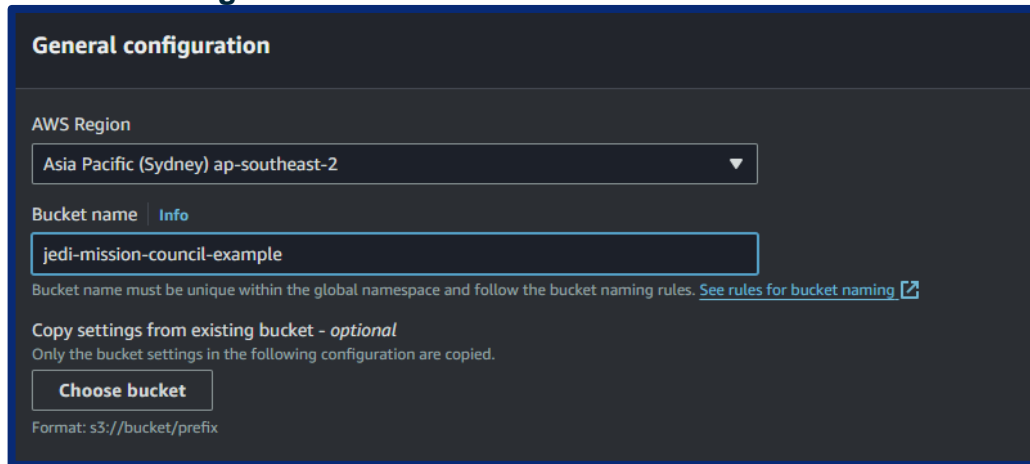
Existing Files

As this service is migrating existing software. We will be using existing docker modules as well as website assets in this document. For the website, we have [this](#) zip file, and for the docker module, we have [this](#) docker file (use this link when using docker).

Migrating to S3

Creation of S3 Bucket

General Configuration



General configuration

AWS Region
Asia Pacific (Sydney) ap-southeast-2

Bucket name [Info](#)
jedi-mission-council-example

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

Copy settings from existing bucket - *optional*
Only the bucket settings in the following configuration are copied.

☐ Choose bucket

Format: s3://bucket/prefix

General configuration of this subnet is reasonably straightforward. We want to create an S3 Bucket based in ap-southeast-2.

From here go into the bucket, upload the zip file provided and extract its contents. Then go into **Properties > Static Website Hosting** and activate it. Have the index.html file be the index file and the error page be the 404.html file.

Permissions and Policy

To prepare our website for public traffic we need to allow external access. Which we achieve by going **Permissions > Block Public Access (bucket settings)** and disable it.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::council-of-the-jedi-www-2314151/*"
    }
  ]
}
```

We then use the following policy to manage permissions for public users. By having the policy PublicReadGetObject, we allow users to access the Jedi Councils website and its associated assets.

Management of CloudFront Distribution

We now need a distribution to manage traffic to this bucket. We do this by creating a CloudFront distribution, with its origin domain as the S3 Bucket we made earlier.

Details		
Distribution domain name d2buuu6lhzacka.cloudfront.net	ARN arn:aws:cloudfront::672129751776:distribution/E37DED9NLXL507	Last modified December 10, 2023 at 9:15:53 AM UTC
Settings		
Description -	Alternate domain names jedicouncil-services.net *.jedicouncil-services.net Custom SSL certificate jedicouncil-services.net	Standard logging Off Cookie logging Off Default root object index.html
Price class Use all edge locations (best performance)	Security policy TLSv1.2_2021	
Supported HTTP versions HTTP/2, HTTP/1.1, HTTP/1.0		

Managing traffic for viewers, we will only enable HTTP/HTTPS as well as GET and POST methods traffic for security reasons. In addition, we should activate WAF to further protect the Jedi Council's assets.

Creating Certificate

Finally, we need a memorable certificate that matches the purpose of the Jedi Council. We chose to do this using AWS Certificate Manager (ACM).

Domain	Status	Renewal status	Type
jedicouncil-services.net	Success	-	CNAME
*.jedicouncil-services.net	Success	-	CNAME

To do this, we changed region to **us-east-1**, requested a certificate with the name, jedicouncil-services.net. This will allow us to use CNAME to direct traffic to our CloudFront distribution.

Route53 Setup

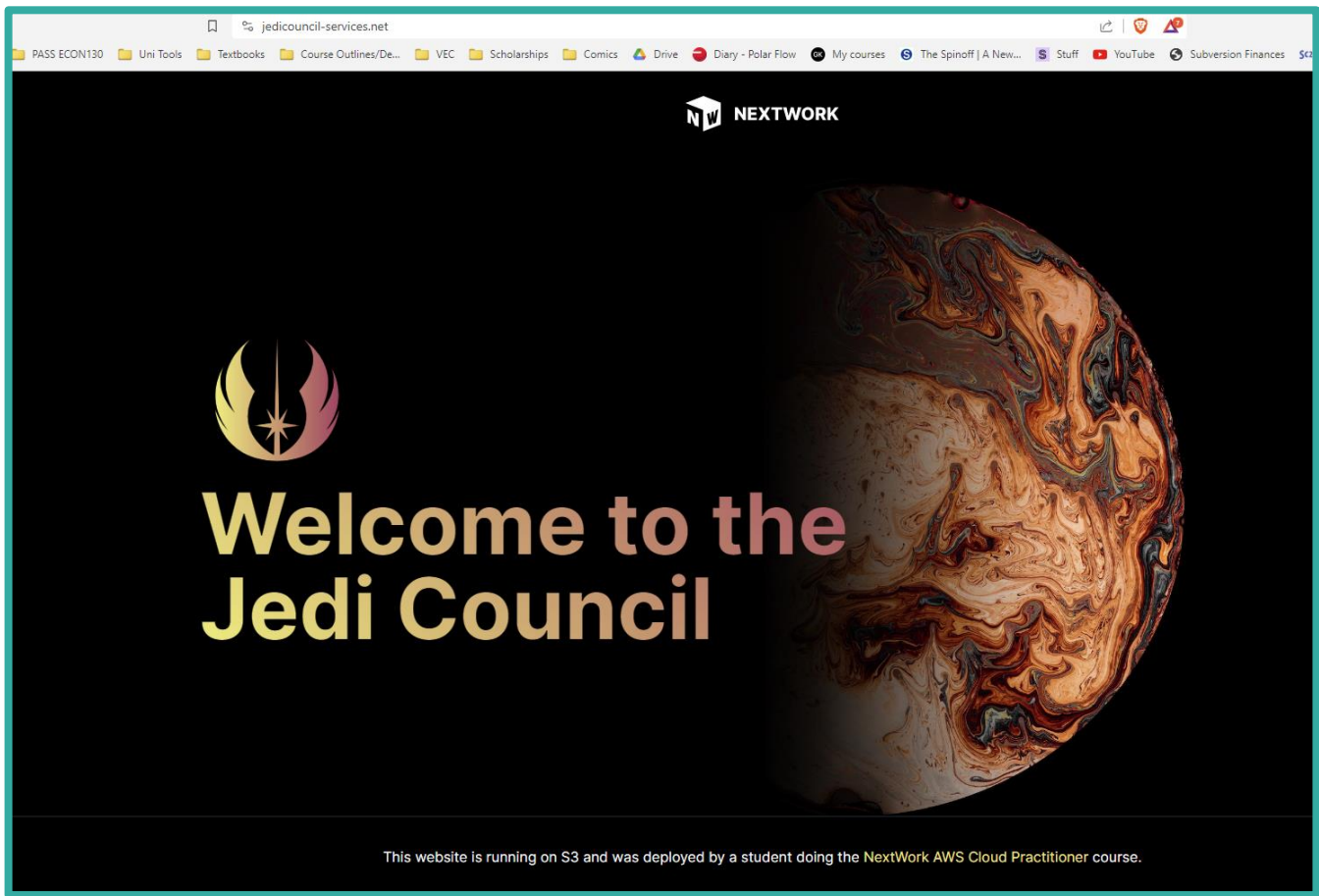
To manage this new domain name, we used Route53 as our DNS management service. We first created a hosted zone with the registered domain name. Ensure that it is a public-hosted-zone.

<input type="checkbox"/>	Record name	Type	Routin...	Differ...	Alias	Value/Route traffic to	TTL (s...
<input type="checkbox"/>	jedicouncil-services.net	SOA	Simple	-	No	ns-877.awsdns-45.net. awsd...	900
<input type="checkbox"/>	jedicouncil-services.net	NS	Simple	-	No	ns-877.awsdns-45.net. ns-281.awsdns-35.com. ns-1335.awsdns-38.org. ns-1940.awsdns-50.co.uk.	172800
<input type="checkbox"/>	jedicouncil-services.net	A	Simple	-	Yes	d2buuu6lhzacka.cloudfront....	-

We then configured the records so that traffic would go to the CloudFront distribution. We did this by changing records: **Records > Create Record**. Here, ensure that the record type is **A** and that we route traffic to the distribution domain name.

We should now have a working website, which has been successfully migrated! If there are any difficulties, ensure that the records are routing traffic correctly (it is always DNS).

Finished Article:



AutoScaling Logging System

Creation of our Virtual Private Cloud (VPC)

To create our VPC, we need nothing special at the beginning. Select **VPC only** and ensure that we have a CIDR of 10.0.0.0/16.

Subnetting

For this service, we want to isolate our instances from public access. To do this, we will create a public and private subnet for **two** availability zones (AZ) in **ap-southeast-2**. We will follow the following IP scheme:

Subnet Name	Availability Zone	Ip address
Public 1	ap-southeast-2a	10.0.0.0/20
Public 2	ap-southeast-2b	10.0.64.0/20
Private 1	ap-southeast-2a	10.0.128.0/20
Private 2	ap-southeast-2b	10.0.192.0/20

Route Tables and Internet Gateways

We now want to configure these subnets to limit the amount of traffic going through them. This means that private subnets should only be able to communicate within the VPC. Public subnets, however, will need to be accessible through the internet.

Create an internet gateway (IGW) and attach it to the VPC. Then create two subnets, one with a rule allowing traffic directed to 0.0.0.0 to the IGW (Public Route Table). The other on should only allow traffic within the VPC (Private Route Table).

Then, to manage the subnets' traffic, go **Subnet Associations > Edit Subnet Associations**. Add the corresponding subnets to their tables. This should mean all your traffic for instances is now sorted!

Mission-Log-Private-rt	rtb-05621efe394d258c5	2 subnets
Mission-Log-Public-rt	rtb-0554734a77bd20beb	2 subnets

Creating Instances

Now, we have created an EC2 instance to start the process of creating both our AutoScaling group as well as managing that relationship with the Relational Database Service.

Configuring Security Groups

To ensure that we are following the principle of least privilege, we will configure a couple security groups. One for general web servers that we can access, and another, more restrictive one for our AutoScaler.

'WebServer' Security Group

Inbound Rules:

Type	Protocol	Port range	Source	Description
SSH	TCP	22	0.0.0.0/0	Allows ssh traffic
HTTP	TCP	80	0.0.0.0/0	Allows inbound http traffic
All ICMP - IPv4	ICMP	All	0.0.0.0/0	Allows ICMP

Outbound Rules:

Type	Protocol	Port range	Destination	Description
All traffic	All	All	0.0.0.0/0	-

'Scaler' Security Group

Inbound Rules:

Type	Protocol	Port range	Source	Description
HTTP	TCP	80	16.0.0.0/16	-

Outbound Rules:

Type	Protocol	Port range	Destination	Description
All traffic	All	All	0.0.0.0/0	-

Note that in future, you could configure the scaler group to only accept incoming traffic from the load balancer for added security.

Creating our First instance

First, create an EC2 (t2-micro) instance, running a linux AMI. Ensure that it is in the resolute VPC and that it auto-assigns its IP. Use the 'WebServer' Security Group we

designed earlier. Once it is running, set up docker on that machine (give it root access as well).

Creation of Relational Database Services (RDS)

Now, create a new RDS service. This should be a free-trier PostgreSQL within our current VPC. We will also need to connect it with a compute resource. This will be the instance that we created earlier. Finally, we create a new security group.

However, we will modify the security group so that it allows traffic to future auto scaled instances. This will be done by adding the 'Scaler' security group to the inbound rules, enabling communication between the two.

Inbound Rules:

Type	▼	Protocol	▼	Port range	▼	Source	▼	Description
PostgreSQL		TCP		5432		sg-0398211956e3fe2...		Rule to allow connections from EC2 instance...
PostgreSQL		TCP		5432		sg-0d754d7ed459ee2...		Allows all connections from all WebServer E...

Deploying Service and Creating AutoScaling Groups

Deploying the Service

Now we need to deploy the docker service. As we already have the docker package, this is only two commands:

```
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:missionLogs@db-mission-logs.cwd9447qak1a.ap-southeast-2.rds.amazonaws.com:5432/jedi-mission-logs public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-projects/jedi-mission-logs:latest
```

We now have a successfully running instance for our mission log service!

Creation of Image and Launch Template

First create an AMI of your service with the running instance. We can then create a launch template for the AutoScaler.

To do this, we will keep similar settings to before (t2-micro) but using the AMI we just made. In addition, we enter the following user data to start the service for any

launched instance:

```
#!/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:missionLogs@db-mission-logs.cwd9447qak1a.ap-southeast-2.rds.amazonaws.com:5432/jedi-mission-logs public.ecr.aws/z7j4c9h0/nextwork/coursework/real-world-projects/jedi-mission-logs:latest
```

Creating the Load Balancer and Target Groups

To create the Load Balancer, configure an Application Load Balancer (ALB). This will need to be **both public subnets** that we configured earlier. We will then create and attach an instance-based target group that operates on port 80.

Creating the AutoScaler

Finally, we configure the autoscaling group. Ensure that this group operates on both **private** subnets and that it has a minimum of 0 instances and a maximum of 5. It will also have a target tracking policy as well as CloudWatch and cost optimization plans. This plan will be the “Control Costs” policy based on CPU utilization.

Once created, we should start to see instances spinning up. However, we won’t be able to access them as they are in the private subnet. To enable access, attach the application load balancer through its target group and use its URL to access the service!

