Arxivism

An academic research tool for knowledge maintenance.

- Student name: **Daniel Manning**
- Student ID: **s3429288**
- Contribution allocation: **All work completed solely by me**.
- Site URL: **http://frontend.eba-vxfjtka5.ap-southeast-2.elasticbeanstalk.com/**
- Repository URL: **https://github.com/dmisdm/cloud-computing-assignment-3**
- Link to this project document (better experience): https://www.notion.so/Arxivism-354411fce4cb466a9b1c01e570f68a43

Signed:

[illegible]

Introduction

As an academic you are likely to experience at least some discomfort due to the unfathomable amount of information that currently exists, and the little time that you have to digest it. In the space of academic tools there are plenty of competitors, each having their own ambitions to solve a unique problem.

Our application, Arxivism, has a mission to improve your academic research workflow by allowing you to search for papers across multiple sources, upload your own to our database, bookmark them, add notes, and collect references. It's primary focus is on the problem of *knowledge maintenance* and assistance using software. Academics read many papers, but diligent note taking and archaic human organisation can only get you so far.

One of the intentions of Arxivism is to be a home for your research journey. You might start with reading a few recommended papers from supervisors or other peers, which is all it takes to start discovering more and gain velocity. There are endless possible tools that could be built to support this, such as a social platform that allows you to find neighbourhood recommendations, or a highly intelligent research paper writing environment (like overleaf.com but much more improved). We have begun our venture into this space with the baseline functionality of searching for articles, bookmarking them, publishing new ones, making notes and comments on articles, and some initial administration analytics into user search queries.

Arxivism's primary intention is to fit into the daily academic research task of reading through papers and keeping track of what you've learned, which is a highly challenging quest to solve using software. It isn't feasible to solve the entire problem in this project given the time constraints, it is however an endeavour to begin. Allowing users the ability to make comments on papers is something unseen within our competitor review, and is something new we offer to real world users.

User Features

- User registration using email, full name, and password, allowing subsequent sign-ins.
- Search for research articles using a free form text query, sourced from both user uploads and arxiv.org
- Publish your own articles and upload a document for them, so that they can be found by other users of Arxivism. Published articles can be seen on the home page.
- Bookmark articles found by searching, which are shown on the home page, and can be unbookmarked.
- View and download any article.
- Make comments on any article.

Admin Features

- Analytics
  - View top 10 most frequent search terms in queries.
  - Re-run the analytics job to get up to date information.

Related Applications

arxiv.org [4]

arXiv is very large research article database, focussed on STEM subjects such as Computer Science. It lets you search for articles, export bibtex citations, dig into references, and much more.

We integrate with their API [8] to power our search engine and bring Arxivism to life.

arxiv-vanity.com [5]

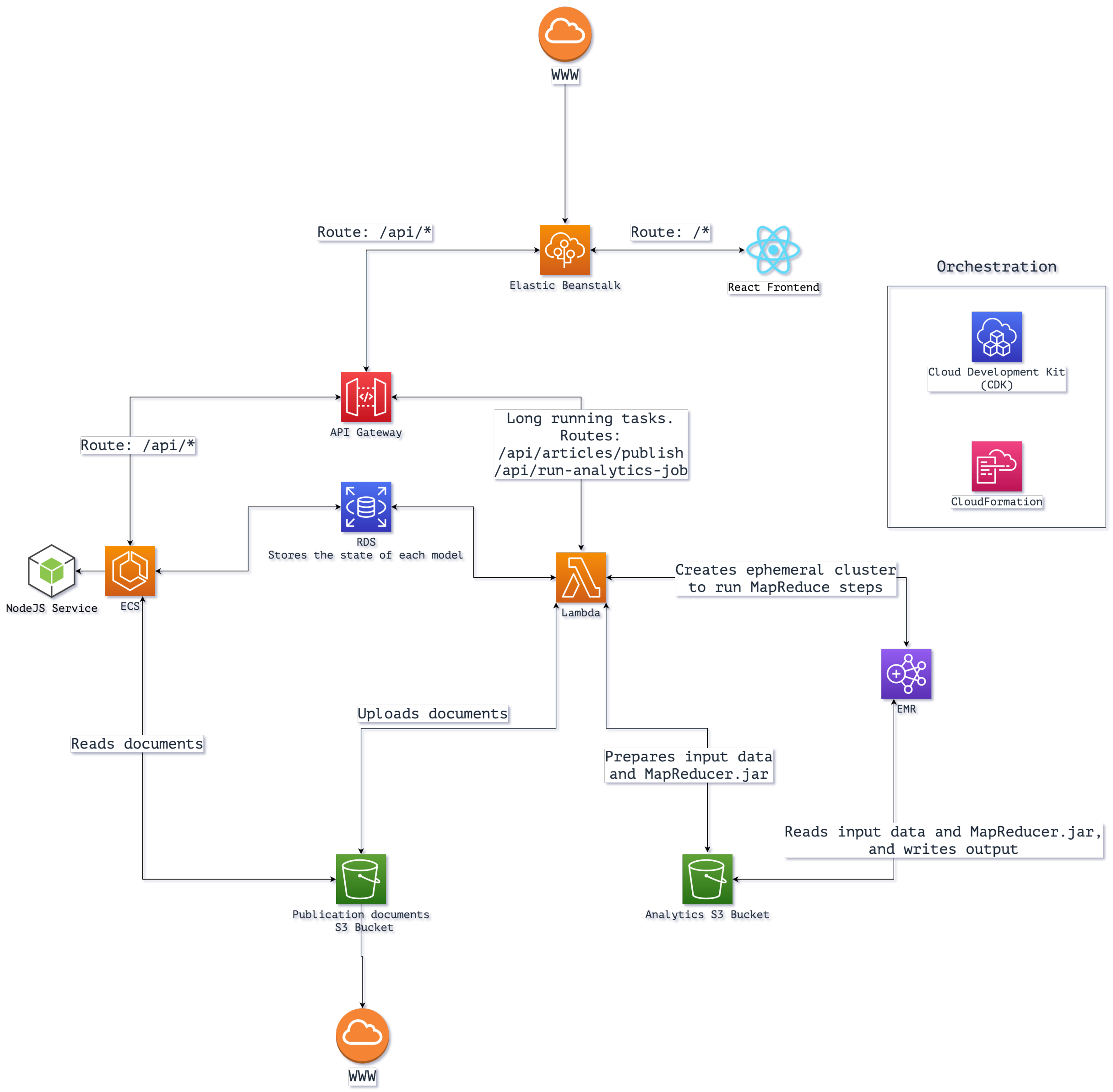
Another tool which solves one crucial issue: the reading experience. PDFs, the format of choice for downloading articles on arXiv if you exclude the manuscript source, don't reflow well, and are therefore highly unresponsive. Luckily, as mentioned, arXiv also hosts manuscript sources (usually TeX or LaTeX) which can be compiled to other formats. arxiv-vanity.com does just that: compiles to HTML so that it can be laid out differently depending on your screen dimensions.

researcher-app.com [6]

A tool which does a very good job at aggregating article sources, and one that notifies you of new papers related to your subscribed categories.

We draw much inspiration from this, but notice that its primary goal is not in machine assisted knowledge maintenance.

System Architecture


WWW
Route: /api/*
Elastic Beanstalk
Route: /*
React Frontend
Orchestration
Cloud Development Kit (CDK)
CloudFormation
API Gateway
Route: /api/*
Long running tasks. Routes: /api/articles/publish /api/run-analytics-job
RDS
Stores the state of each model
NodeJS Service
ECS
Lambda
Creates ephemeral cluster to run MapReduce steps
EMR
Uploads documents
Reads documents
Prepares input data and MapReducer.jar
Reads input data and MapReducer.jar, and writes output
Publication documents S3 Bucket
Analytics S3 Bucket
WWW

Architecture of the entire system

Critical Path

1. Users enter the application through an Elastic Beanstalk server, which hosts a static asset React frontend on routes `/*` using Vite.JS [7]. On routes `/api/*` which takes precedence over the frontend, requests are proxied to an API Gateway, which is also served using Vite.JS. This React app calls HTTP endpoints on `/api/*` to carry out functionality.

   `vite.config.js`:

```
// backendUrl is defined at runtime and decided by the infrastructure.
 // It will effectively end up being the endpoint for the API Gateway.
 const backendUrl = process.env.BACKEND_URL || "http://localhost:8080";

 export default defineConfig({
   //...
   server: {
     proxy: {
       "/api": backendUrl,
     },
   },
 });
```

2. API Gateway then either forwards requests to an NodeJS server hosted using ECS, or to a Lambda Function depending on the resource (route).
   - Lambda Function endpoints.
     These are long running tasks that are purposely hosted using Lambda to prevent the main NodeJS backend from being blocked by long running tasks.
     - `/api/articles/publish` : The endpoint that supports uploading of publications. Because these files can be of arbitrary size and need to be uploaded to S3, it is assumed to be a long running task that should be dynamically parallelised using Lambda.
     - `/api/run-analytics-job`: The endpoint that creates an ephemeral EMR cluster (one that is terminated after completion) and a job to carry out a MapReduce function which calculates term frequencies across all search queries, to then be displayed within the app at `/analytics` by reading the output files written to the analytics S3 bucket by EMR.
3. Everything else on `/api/*` is forwarded to the NodeJS server on ECS, hosting the functionality mentioned in the introduction such as `/api/search`, `/api/articles/bookmarks` and more.
   - This includes communicating with an RDS (PostgreSQL engine) instance for stateful queries and mutations.

AWS Services

In the following sections we describe how each AWS service was utilised and configured to support our application.

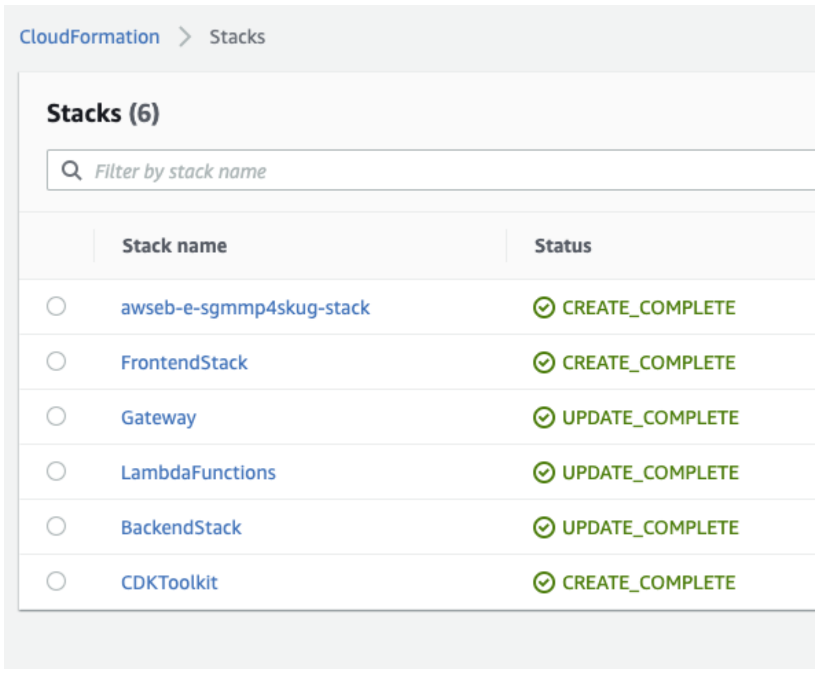
CloudFormation and Cloud Development Kit (CDK)

We utilise the CDK to declaratively write and automate every part of our infrastructure using TypeScript.

This snippet shows the top level CDK code which orchestrates 4 stacks: Backend, Lambda, Gateway and Frontend. Splitting out stacks like this assists with separating concerns.

```
const envOptions = {
  env: {
    account: "163565994931",
    region: "ap-southeast-2",
  },
};
const app = new cdk.App();
const backendStack = new BackendStack(app, "BackendStack", {}, envOptions);
const backendLoadBalancerEndpoint =
  "http://" + backendStack.alb.loadBalancerDnsName;

const lambdaStack = new LambdaFunctionsStack(
  app,
  "LambdaFunctions",
  {
    vpc: backendStack.vpc,
    databaseSecretName: backendStack.databaseSecretName,
    publicationsBucket: backendStack.publicationsBucket,
    serviceSecurityGroup: backendStack.serviceSecurityGroup,
    analyticsBucket: backendStack.analyticsBucket,
  },
  envOptions
);

const gateway = new GatewayStack(
  app,
  "Gateway",
  {
    backendLambda: lambdaStack.function,
    backendLoadBalancerEndpoint,
  },
  envOptions
);

new FrontendStack(
  app,
  "FrontendStack",
  { backendUrl: gateway.api.url! },
  envOptions
);
```

Deploying these stacks results in this:



Stack name	Status
awseb-e-sgmmp4skug-stack	CREATE_COMPLETE
FrontendStack	CREATE_COMPLETE
Gateway	UPDATE_COMPLETE
LambdaFunctions	UPDATE_COMPLETE
BackendStack	UPDATE_COMPLETE
CDKToolkit	CREATE_COMPLETE

Elastic Beanstalk

Elastic Beanstalk's responsibility is for hosting the frontend application, and proxying requests to API Gateway (which contains further proxies to other services). It exposes the single entrypoint for users into our application.

We deploy our frontend in a separate stack, containing primarily an Elastic Beanstalk construct, configured to deploy a zip (built in a previous step) of our codebase.

Below is the main snippet (commented) of our CDK code that deploys the frontend stack; the CDK code that supports the configuration and deployment of the frontend using Elastic Beanstalk.

```
export class FrontendStack extends cdk.Stack {
constructor(
  scope: cdk.Construct,
  id: string,
  dependencies: { backendUrl: string },
  props?: cdk.StackProps
) {

  super(scope, id, props);

  exec("./bundle.sh", {
    cwd: fromRoot("."),
  });
  // The main EB application which other resources are added to.
  const frontend = new elasticbeanstalk.CfnApplication(this, "Frontend", {
    applicationName: "Frontend",
  });

  // This construct allows us to automatically upload a zip file of our code, and reference it in subsequent constructs.
  const frontendAssets = new s3assets.Asset(this, "FrontendAssets", {
    path: fromRoot("bundle.zip"),
  });

  const platform = this.node.tryGetContext("platform");

  // The description of where to find code assets (in this case, it is a zip file on S3)
  const latestFrontendVersion = new elasticbeanstalk.CfnApplicationVersion(
    this,
    "LatestFrontendVersion",
    {
      applicationName: "Frontend",
      sourceBundle: {
        s3Bucket: frontendAssets.s3BucketName,
        s3Key: frontendAssets.s3ObjectKey,
      },
    }
  );

  // The IAM execution role. No special permissions are required for this role, as it isn't accessing any other private resources.
  const frontendRole = new iam.Role(this, `FrontendRole`, {
    assumedBy: new iam.ServicePrincipal("ec2.amazonaws.com"),
  });

  frontendRole.addManagedPolicy(
    iam.ManagedPolicy.fromAwsManagedPolicyName("AWSElasticBeanstalkWebTier")
  );

  const profileName = `FrontendEBProfile`;
  const instanceProfile = new iam.CfnInstanceProfile(this, profileName, {
    instanceProfileName: profileName,
    roles: [frontendRole.roleName],
  });

  // The EB environment which explains how to construct EC2 instances and run our application.
  const frontendEnvironment = new elasticbeanstalk.CfnEnvironment(
    this,
    "FrontendEnvironment",
    {
      applicationName: "Frontend",
      platformArn: platform,
      versionLabel: latestFrontendVersion.ref,
      solutionStackName: "64bit Amazon Linux 2 v5.4.0 running Node.js 14",
      environmentName: "Frontend",
      optionSettings: [
        {
          namespace: "aws:autoscaling:launchconfiguration",
          optionName: "IamInstanceProfile",
          value: profileName,
        },
        {
          namespace: "aws:autoscaling:launchconfiguration",
          optionName: "InstanceType",
          value: "t3.small",
        },

        {
          namespace: "aws:elasticbeanstalk:application:environment",
          optionName: "BACKEND_URL",
          value: dependencies.backendUrl,
        },
      ],
    }
  );

  latestFrontendVersion.addDependsOn(frontend);

  frontendEnvironment.addDependsOn(latestFrontendVersion);

  frontendEnvironment.addDependsOn(frontend);

  }
}
```

This snippet shows the processes that are run by Elastic Beanstalk within it's deployed templated EC2 instances. This defines how the web server is run (`Procfile`).

```
web: yarn start
```

The `package.json` file: the start script is what is run when `yarn start` is called.

```
"scripts": {
    "start": "yarn workspace web dev --host --port ${PORT:-8080}",
    "bundle": "zip -r frontendAssets.zip ./ -x .git\\* \\*node_modules\\* .yarn/unplugged\\* .yarn/cache\\* ops\\*"
  },
```

The webserver started using `yarn start` serves frontend assets and proxies requests on `/api/*` to API Gateway, as specified by `dependencies.backendURL` which is passed in

Elastic Container Service (ECS) and RDS

ECS is used to serve the larger half portion of the backend, which is a NodeJS server.

The NodeJS server is first dockerised:

```
FROM node:alpine as builder
ARG NODE_ENV=production
ENV NODE_ENV=${NODE_ENV}

WORKDIR /usr/src/app

COPY package.json .yarnrc.yml yarn.lock ./
COPY .yarn .yarn
COPY server server
COPY prisma-client prisma-client
COPY analytics analytics
WORKDIR /usr/src/app/server
RUN ls -la
RUN yarn workspaces focus && yarn build && yarn workspaces focus --production && yarn dedupe && yarn cache clean --all

CMD ["./run-in-docker.sh"]
```

It is then uploaded to ECR (AWS's docker image registry) to be used in a task definition within an ECS service.

RDS is used to support stateful operations such as:

- User registration
- Login, authentication, and authorisation
- Listing, adding, and removing bookmarks
- Listing and publishing articles (executed using a Lambda function, not on ECS)
- Searching user published articles
- Commenting
- User Events (at the moment, only Search queries are logged)

The models used within the application are defined and managed using an ORM (object relational mapper) called Prisma [10]. See the section below for the entire schema.

Below is the CDK code that describes the whole backend stack's configuration. Observe the comments to understand the purpose of each construct.

```
export class BackendStack extends cdk.Stack {
  alb: ApplicationLoadBalancer;
  vpc: Vpc;
  serviceSecurityGroup: SecurityGroup;
  publicationsBucket: Bucket;
  analyticsBucket: Bucket;
  databaseSecretName: string;
  constructor(
    scope: cdk.Construct,
    id: string,
    { backendImageTag }: { backendImageTag?: string },
    props?: cdk.StackProps
  ) {
    super(scope, id, props);
    const internalTrafficSubnetName = "internal-traffic";
    const fromInternetSubnetName = "from-internet";

    // The VPC that will contain the ECS service and RDS.
    const vpc = new ec2.Vpc(this, "MainVPC", {
      maxAzs: 2,
      subnetConfiguration: [
        {
          cidrMask: 24,
          name: fromInternetSubnetName,
          subnetType: SubnetType.PUBLIC,
        },
        {
          cidrMask: 24,
          name: internalTrafficSubnetName,
          subnetType: SubnetType.PRIVATE,
        },
      ],
    });

    this.vpc = vpc;

    //The cheapest Postgres RDS instance.
    const database = new rds.DatabaseInstance(this, "database", {
      vpc,
      instanceType: new InstanceType("t3.micro"),
      allocatedStorage: 20,
      storageType: rds.StorageType.STANDARD,
      backupRetention: Duration.days(0),
      engine: DatabaseInstanceEngine.postgres({
        version: rds.PostgresEngineVersion.VER_13,
      }),
      vpcSubnets: vpc.selectSubnets({
```

```
      subnetType: SubnetType.PRIVATE,
    }),
  });

  // Here we select only a few parts of the database secret to extract it's name.
  // Unfortunately, database.secret.secretName contains an unneeded suffix that causes errors otherwise.
  const secretNameParts = Fn.split("-", database.secret?.secretName!);
  const secretNameWithoutSuffix = Fn.join("-", [
    Fn.select(0, secretNameParts),
    Fn.select(1, secretNameParts),
  ]);
  this.databaseSecretName = secretNameWithoutSuffix;

  // The S3 bucket to support publication uploading
  const publicationsBucket = new s3.Bucket(this, "Publications", {
    bucketName: "cloud-computing-assignment-3-publications",
    accessControl: BucketAccessControl.PUBLIC_READ,
    publicReadAccess: true,
    versioned: false,
    removalPolicy: cdk.RemovalPolicy.DESTROY,
  });

  // The S3 bucket to support analytics
  const analyticsBucket = new s3.Bucket(this, "EMRAnalytics", {
    bucketName: "cloud-computing-assignment-3-analytics",
    versioned: false,
    removalPolicy: cdk.RemovalPolicy.DESTROY,
  });

  this.analyticsBucket = analyticsBucket;

  this.publicationsBucket = publicationsBucket;

  const cluster = new ecs.Cluster(this, "BackendCluster", {
    vpc: vpc,
  });

  // The ECS task role. We need the base policy, plus the ability to get the databse secret, and to execute EMR tasks at runtime.
  const taskRole = new iam.Role(this, "NodeJSServerTaskRole", {
    roleName: "NodeJSServerTaskRole",
    assumedBy: new iam.ServicePrincipal("ecs-tasks.amazonaws.com"),
    managedPolicies: [
      iam.ManagedPolicy.fromAwsManagedPolicyName(
        "service-role/AmazonECSTaskExecutionRolePolicy"
      ),
      iam.ManagedPolicy.fromAwsManagedPolicyName(
        "AmazonEMRFullAccessPolicy_v2"
      ),
    ],
  });

  taskRole.addToPolicy(
    new iam.PolicyStatement({
      effect: iam.Effect.ALLOW,
      resources: ["*"],
      actions: ["secretsmanager:GetSecretValue", "elasticmapreduce:*"],
    })
  );

  const nodejsServerTask = new ecs.FargateTaskDefinition(
    this,
    "NodeJSServer",
    {
      cpu: 512,
      memoryLimitMiB: 1024,
      taskRole,
    }
  );

  // This construct will build the specified Dockerfile and upload it to ECR automatically.
  const serverImage = EcrImage.fromAsset(fromRoot("."), {
    file: "server/Dockerfile",
  });

  const nodejsServerContainer = nodejsServerTask.addContainer(
    "nodejs-server",
    {
      image: serverImage,
      environment: {
        NODE_ENV: "production",
        PORT: "80",
        ARTICLES_BUCKET: publicationsBucket.bucketName,
        ANALYTICS_BUCKET: analyticsBucket.bucketName,
      },
      secrets: {
        POSTGRES_SECRET_JSON: ecs.Secret.fromSecretsManager(database.secret!),
      },
      logging: ecs.LogDriver.awsLogs({ streamPrefix: "nodejs-service" }),
    }
  );

  // Grant additional permissions so that the buckets and database secret can be read from/written to.
  publicationsBucket.grantReadWrite(taskRole);
  analyticsBucket.grantReadWrite(taskRole);
  database.secret!.grantRead(taskRole);

  nodejsServerContainer.addPortMappings({
    containerPort: 80,
  });

  // The security group attached to the ECS service.
  // We need this so that we can tell the database to allow traffic from it - the Nodejs server.
  const serviceSecurityGroup = new ec2.SecurityGroup(
    this,
    "ServiceSecurityGroup",
    {
      vpc,
      allowAllOutbound: true,
    }
  );

  this.serviceSecurityGroup = serviceSecurityGroup;

  // Allow the nodejs server to communicate with the database.
  database.connections.allowFrom(serviceSecurityGroup, ec2.Port.tcp(5432));

  const service = new ecs.FargateService(this, "BackendService", {
    taskDefinition: nodejsServerTask,
    cluster,
    vpcSubnets: vpc.selectSubnets({
      subnetType: SubnetType.PRIVATE,
    }),
    securityGroups: [serviceSecurityGroup],
  });

  const albSecurityGroup = new ec2.SecurityGroup(
    this,
    "ServerALBSecurityGroup",
    { vpc }
  );

  // Expose the ECS service's load balancer to the public.
  albSecurityGroup.connections.allowFromAnyIpv4(ec2.Port.tcp(80));
  albSecurityGroup.connections.allowFromAnyIpv4(ec2.Port.tcp(443));

  // The ECS target group that tells the load balancer where to direct traffic to, and what nodes it can load balance over.
  // In this case, we only have a single instance, so there is no load balancing, but there is still an ALB that exposes the service.
  const targetGroup = new elb.ApplicationTargetGroup(
    this,
    "ServerTargetGroup",
    {
      vpc,
      port: 80,
      targets: [service],
      deregistrationDelay: Duration.seconds(1),
      healthCheck: {
        path: "/api/health",
        timeout: Duration.seconds(30),
        interval: Duration.seconds(45),
        unhealthyThresholdCount: 3,
      },
    }
  );

  const alb = new elb.ApplicationLoadBalancer(this, "BackendALB", {
    vpc,
      securityGroup: albSecurityGroup,
      internetFacing: true,
      vpcSubnets: vpc.selectSubnets({
        subnetType: SubnetType.PUBLIC,
      }),
    });

    const listener = new elb.ApplicationListener(this, "ServerListener", {
      loadBalancer: alb,
      open: true,
      port: 80,
    });
    listener.addTargetGroups("ServerTargetGroups", {
      targetGroups: [targetGroup],
    });

    // Accept incoming connections from the ALB
    serviceSecurityGroup.connections.allowFrom(alb, ec2.Port.tcp(80));

    this.alb = alb;
  }
}
```

Lambda

A single Lambda function is used in our application to support long running tasks. So far, the two tasks allocated to use Lambda are:

- `/api/run-analytics-job` - The endpoint that runs an EMR job and updates analytics.
- `/api/articles/publish` - The endpoint that allows users to upload articles.

Below is the CDK code that configures this Lambda. See the comments for further understanding.

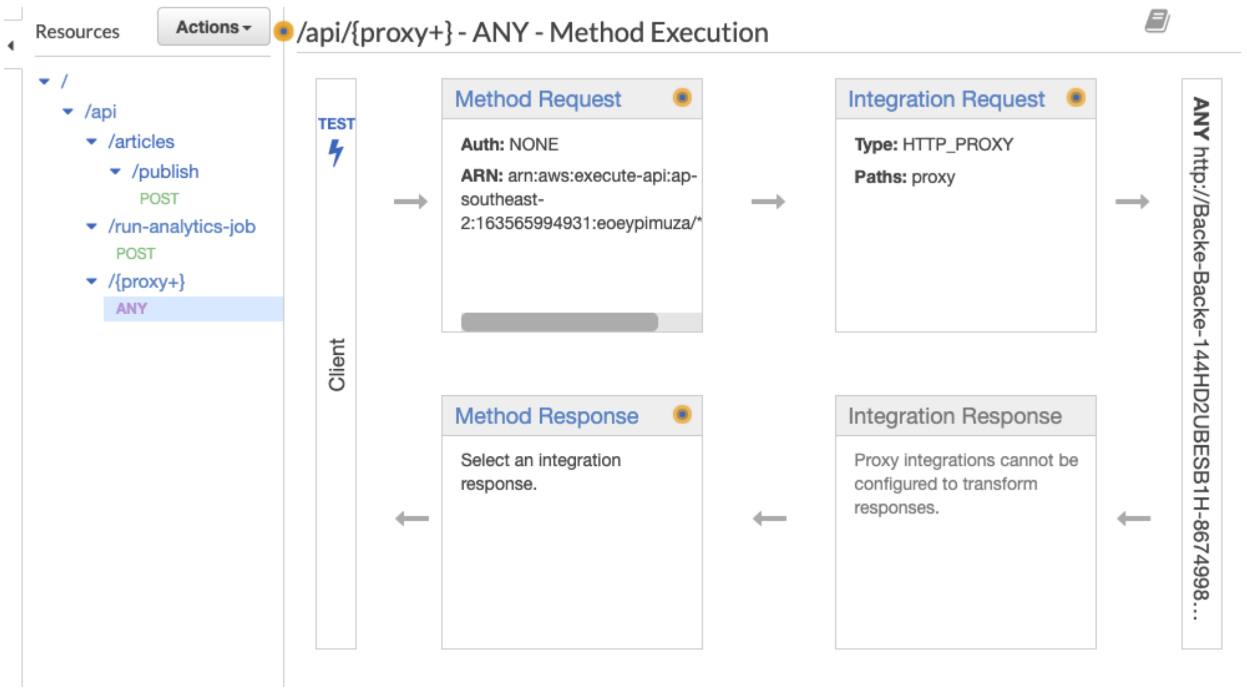
Note: this stack does not expose the Lambda. It is exposed in the subsequent Gateway stack.

```
export class LambdaFunctionsStack extends cdk.Stack {
  function: lambda.Function;
  constructor(
    scope: cdk.Construct,
    id: string,
    // Here we accept dependencies:
    // The backend VPC
    // The backend service security group (we also use this for the lambda so that it has the same access within the VPC)
    // Publications and analytics bucket.
    // Database secret
    {
      vpc,
      serviceSecurityGroup,
      publicationsBucket,
      analyticsBucket,
      databaseSecretName,
    }: {
      vpc: Vpc;
      serviceSecurityGroup: SecurityGroup;
      publicationsBucket: Bucket;
      analyticsBucket: Bucket;
      databaseSecretName: string;
    },
    props?: cdk.StackProps
  ) {
    super(scope, id, props);

    // This lambda requires the same permissions as the ECS service
    // Base execution role, EMR, database, and S3 bucket access
    const lambdaRole = new iam.Role(this, "UploaderLambdaRole", {
      roleName: "UploaderLambdaRole",
      assumedBy: new iam.ServicePrincipal("lambda.amazonaws.com"),
      managedPolicies: [
        iam.ManagedPolicy.fromAwsManagedPolicyName(
          "service-role/AWSLambdaBasicExecutionRole"
        ),
        iam.ManagedPolicy.fromAwsManagedPolicyName(
          "service-role/AWSLambdaVPCAccessExecutionRole"
        ),
        iam.ManagedPolicy.fromAwsManagedPolicyName(
          "AmazonEMRFullAccessPolicy_v2"
        ),
      ],
    });

    // Give the lambda access to EMR and secrets manager (for the database secret)
    lambdaRole.addToPolicy(
      new iam.PolicyStatement({
        effect: iam.Effect.ALLOW,
        resources: ["*"],
        actions: ["secretsmanager:GetSecretValue", "elasticmapreduce:*"],
      })
    );

    analyticsBucket.grantReadWrite(lambdaRole);
    publicationsBucket.grantReadWrite(lambdaRole);

    // We are deploying a Docker container Lambda function.
    // This construct builds a Dockerfile and uploads it to ECR automatically.
    const lambdaCode = lambda.DockerImageCode.fromImageAsset(fromRoot("."), {
      file: "server/lambda.Dockerfile",
    });

    // The main lambda definition which connects the other constructs.
    const uploaderLambda = new lambda.DockerImageFunction(
      this,
      "UploaderLambda",
      {
        functionName: "BackendLambda",
        memorySize: 512,
        vpc,
        vpcSubnets: vpc.selectSubnets({
          subnetType: SubnetType.PRIVATE,
        }),
        role: lambdaRole,
        securityGroups: [serviceSecurityGroup],
        code: lambdaCode,
        environment: {
          NODE_ENV: "production",
          ARTICLES_BUCKET: publicationsBucket.bucketName,
          POSTGRES_SECRET_ARN: databaseSecretName,
          ANALYTICS_BUCKET: analyticsBucket.bucketName,
        },
        timeout: Duration.seconds(15),
      }
    );

    this.function = uploaderLambda;
  }
}
```

API Gateway

API Gateway exposes the Lambda and the ECS service as a unified one, and allows us to direct traffic in a practical manner.

The configuration of our gateway looks like:


Resources
Actions
/api/{proxy+} - ANY - Method Execution
/
/api
/articles
/publish
POST
/run-analytics-job
POST
/{proxy+}
ANY
TEST
Client
Method Request
Auth: NONE
ARN: arn:aws:execute-api:ap-southeast-2:163565994931:eoeypimuza/*
Integration Request
Type: HTTP_PROXY
Paths: proxy
Method Response
Select an integration response.
Integration Response
Proxy integrations cannot be configured to transform responses.
ANY http://Backe-Backe-144HD2UBESB1H-8674998...

Two leaf resources are configured to use the Lambda as mentioned above, and anything else on `/api/*` is directed to our ECS NodeJS service.

The commented CDK code for the gateway stack:

```
export class GatewayStack extends cdk.Stack {
  api: RestApi;
  constructor(
    scope: cdk.Construct,
    id: string,
    {
      // We depend on the backend lambda, and the ECS load balancer endpoint, for configuring a unified Gateway
      backendLambda,
      backendLoadBalancerEndpoint,
    }: { backendLambda: Function; backendLoadBalancerEndpoint: string },
    props?: cdk.StackProps
  ) {
    super(scope, id, props);

    // The root api gateway construct
    const api = new apigateway.RestApi(this, "BackendFederatedApi", {
      restApiName: "Backend",
      description:
        "The API that federates every other backend/api within our stack into one",
    });

    this.api = api;

    // The integration that describes how to forward requests to the Lambda.
    const lambdaIntegration = new apigateway.LambdaIntegration(backendLambda, {
      proxy: true,
    });

    // The integration that describes how to forward requests to our ECS service.
    // Nothing is specific to ECS here, it is more abstractly defined as a HTTP proxy/integration
    const fargateServiceIntegration = new apigateway.HttpIntegration(
      `${backendLoadBalancerEndpoint}/api/{proxy}`,
      {
        proxy: true,
        httpMethod: "ANY",
        options: {
          requestParameters: {
            "integration.request.path.proxy": "method.request.path.proxy",
          },
        },
      }
    );

    // The /api resource
    const rootApiResource = api.root.addResource("api");


    // This enables all traffic to forward to the ECS service on `/api/{proxy+}` as a fallback.
    const ecsServiceProxyMethod = rootApiResource.addProxy({
      anyMethod: true,
      defaultIntegration: fargateServiceIntegration,
      defaultMethodOptions: {
        requestParameters: {
          "method.request.path.proxy": true,
        },
      },
    });

    // The /api/articles/publish resource
    const publishArticleResource = rootApiResource
      .addResource("articles")
      .addResource("publish");

    // The /api/articles/publish POST method, that integrates with the backend lambda.
    const publishArticleMethod = publishArticleResource.addMethod(
      "POST",
      lambdaIntegration
    );

    // Similarly, this allocates /api/run-analytics-job to use the lambda.
    const analyticsJob = rootApiResource.addResource("run-analytics-job");
    analyticsJob.addMethod("POST", lambdaIntegration);
  }
}
```

Elastic MapReducer (EMR)

We utilise EMR for producing analytics in a scalable manner. At this point, the only analytics question we are answering is: what are the top 10 most frequent search terms within article search queries?

We use a combination of our Lambda Function, an S3 bucket, and an EMR cluster with a MapReduce job in the following way for the `/api/run-analytics-job` endpoint:

1. List all search event queries that are logged within Postgres.
2. Write them all to a single file, and upload to the analytics S3 bucket. This is the input file to our MapReduce task.
3. A JVM application (written in Kotlin for Hadoop) is also uploaded to S3
    1. The mapper takes each input line (which is a search query) and tokenises it, writing a new line consisting of `<word> 1`
    2. The reducer then treats each `<word>` as a key, and sums each entry produced by the mapper.
    3. With this, we end up with aggregated (reduced) entries of `<word> <totalCount>`
4. This Hadoop MapReducer then writes the results to an output file on S3.
5. This is then served as JSON on the ECS backend API on `/api/most-frequent-search-terms` and displayed on the `/analytics` web page.

Rather than use the CDK for EMR, we utilise the NodeJS client SDK and execute this at runtime within the Lambda function.

```
export async function run({
  awsRegion,
  bucketName,
  inputObjectKey,
  outputObjectKey,
}: {
  awsRegion: string;
  bucketName: string;
  inputObjectKey: string;
  outputObjectKey: string;
}) {
  const jarName = "mapreducer.jar";
  const s3Client = new s3.S3Client({ region: awsRegion });
  const putCommand = new s3.PutObjectCommand({
    Key: "mapreducer.jar",
    Bucket: bucketName,
    Body: jarContents,
  });
  console.log("Uploading jar..");
  await s3Client.send(putCommand);
  console.log("Successfully uploaded jar");
  console.log("Setting up EMR");

  const emrClient = new emr.EMR({
    region: awsRegion,
  });
  const result = await emrClient.runJobFlow({
    Name: "ComputeAnalytics",
    ServiceRole: "EMR_DefaultRole",
    JobFlowRole: "EMR_EC2_DefaultRole",
    ScaleDownBehavior: "TERMINATE_AT_TASK_COMPLETION",
    ReleaseLabel: "emr-5.33.0",
    LogUri: `s3://${bucketName}/logs`,
    Instances: {
      MasterInstanceType: "m4.large",
      Ec2KeyName: "Default",
      InstanceCount: 1,
    },

    Steps: [
      {
        Name: "step",
        ActionOnFailure: "TERMINATE_CLUSTER",

        HadoopJarStep: {
          Jar: `s3://${bucketName}/${jarName}`,
          MainClass: "arxivism.WordCount",
          Args: [
            "arxivism.WordCount",
            `s3://${bucketName}/${inputObjectKey}`,
            `s3://${bucketName}/${outputObjectKey}`,
          ],
        },
      },
    ],
  });
}
```

S3

We've made allusions to S3 throughout the use of other services. For clarity, we re-iterate that S3 is being used for 2 buckets: publication documents and and analytics for EMR.

```
// The S3 bucket to support publication uploading
const publicationsBucket = new s3.Bucket(this, "Publications", {
  bucketName: "cloud-computing-assignment-3-publications",
  accessControl: BucketAccessControl.PUBLIC_READ,
  publicReadAccess: true,
  versioned: false,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});

// The S3 bucket to support analytics
const analyticsBucket = new s3.Bucket(this, "EMRAnalytics", {
  bucketName: "cloud-computing-assignment-3-analytics",
  versioned: false,
  removalPolicy: cdk.RemovalPolicy.DESTROY,
});
```

As noted, the publications bucket has public read access enabled, allowing users to find and download them through search.

External Services

As mentioned, the arXiv public search API is utilised to source research articles. This is one single endpoint exposed by arXiv that returns search results as XML, allowing you to specify a search query along with paging options (max results and start offset).

An example search query result can be seen here:

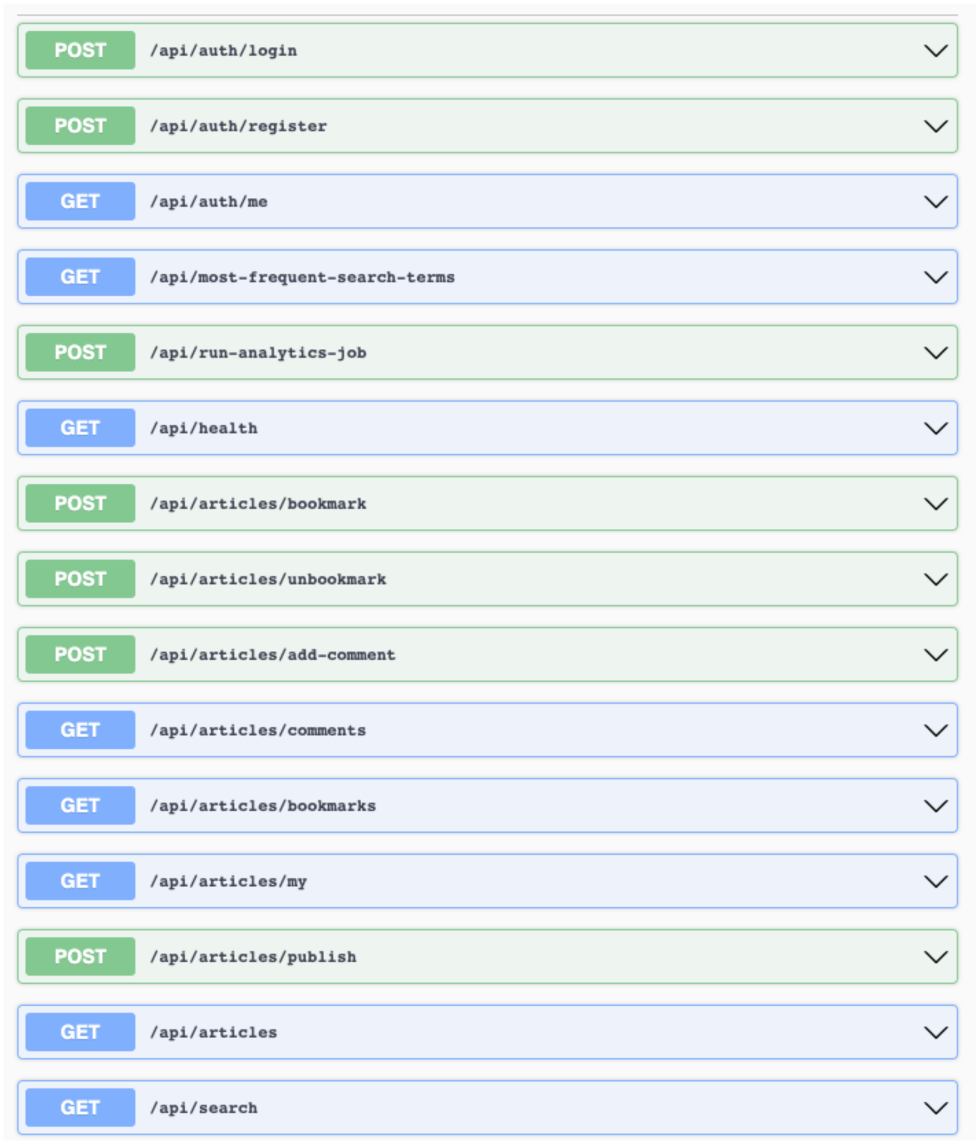
http://export.arxiv.org/api/query?search_query=deep_learning

A single search result within the response looks like:

```
<entry>
    <id>http://arxiv.org/abs/1805.08355v1</id>
    <updated>2018-05-22T02:12:33Z</updated>
    <published>2018-05-22T02:12:33Z</published>
    <title>Opening the black box of deep learning</title>
    <summary>  The great success of deep learning shows that its technology contains
profound truth, and understanding its internal mechanism not only has important
implications for the development of its technology and effective application in
various fields, but also provides meaningful insights into the understanding of
human brain mechanism. At present, most of the theoretical research on deep
learning is based on mathematics. This dissertation proposes that the neural
network of deep learning is a physical system, examines deep learning from
three different perspectives: microscopic, macroscopic, and physical world
views, answers multiple theoretical puzzles in deep learning by using physics
principles. For example, from the perspective of quantum mechanics and
statistical physics, this dissertation presents the calculation methods for
convolution calculation, pooling, normalization, and Restricted Boltzmann
Machine, as well as the selection of cost functions, explains why deep learning
must be deep, what characteristics are learned in deep learning, why
Convolutional Neural Networks do not have to be trained layer by layer, and the
limitations of deep learning, etc., and proposes the theoretical direction and
basis for the further development of deep learning now and in the future. The
brilliance of physics flashes in deep learning, we try to establish the deep
learning technology based on the scientific theory of physics.
</summary>
  <author>
    <name>Dian Lei</name>
  </author>
  <author>
    <name>Xiaoxiao Chen</name>
  </author>
  <author>
    <name>Jianfei Zhao</name>
  </author>
  <link href="http://arxiv.org/abs/1805.08355v1" rel="alternate" type="text/html"/>
  <link title="pdf" href="http://arxiv.org/pdf/1805.08355v1" rel="related" type="application/pdf"/>
  <arxiv:primary_category xmlns:arxiv="http://arxiv.org/schemas/atom" term="cs.LG" scheme="http://arxiv.org/schemas/atom"/>
  <category term="cs.LG" scheme="http://arxiv.org/schemas/atom"/>
  <category term="stat.ML" scheme="http://arxiv.org/schemas/atom"/>
</entry>
```

```
```

API Overview



```
POST /api/auth/login
POST /api/auth/register
GET /api/auth/me
GET /api/most-frequent-search-terms
POST /api/run-analytics-job
GET /api/health
POST /api/articles/bookmark
POST /api/articles/unbookmark
POST /api/articles/add-comment
GET /api/articles/comments
GET /api/articles/bookmarks
GET /api/articles/my
POST /api/articles/publish
GET /api/articles
GET /api/search
```

Database Schema (Prisma [10])

```
enum UserRole {
  Admin
  Basic
}

model User {
  id        Int        @id @default(autoincrement())
  email     String     @unique
  name      String
  password  String
  roles     UserRole[]
  authored  Article[]  //Articles relation
  bookmarks Bookmark[] //Bookmarks relation
  comments  Comment[]  //Comments relation
  createdAt DateTime   @default(now())
  updatedAt DateTime   @default(now()) @updatedAt
}

model Bookmark {
  id        Int      @id @default(autoincrement())
  user      User     @relation(fields: [userId], references: [id])
  userId    Int
  article   Article  @relation(fields: [articleId], references: [id])
  articleId String
  createdAt DateTime @default(now())
  updatedAt DateTime @default(now()) @updatedAt

  @@unique([userId, articleId])
}

enum ArticleSource {
  User
  Arxiv
}

model Article {
  id           String        @id @default(cuid())
  source       ArticleSource
  authors      User[]
  documentUrl  String
  title        String
  summary      String
  categories   String[]
  bookmarks    Bookmark[]    //Relation containing all the bookmarks an article has
  comments     Comment[]
  createdAt    DateTime      @default(now())
  updatedAt    DateTime      @default(now()) @updatedAt
  arxivArticle ArxivArticle?
}

model ArxivArticle {
  id         String   @id @default(cuid())
  title      String
  summary    String
  categories String[]
  authors    String[]
  published  DateTime
  updated    DateTime
  article    Article  @relation(fields: [articleId], references: [id])
  articleId  String
}

model Comment {
  id        Int      @id @default(autoincrement())
  text      String
  author    User     @relation(fields: [authorId], references: [id])
  authorId  Int
  article   Article  @relation(fields: [articleId], references: [id])
  articleId String
  createdAt DateTime @default(now())
  updatedAt DateTime @default(now()) @updatedAt
}

model Event {
  id         Int      @id @default(autoincrement())
  type       String
  parameters Json
  date       DateTime @default(now())
}
```

Developer Manual

Our whole infrastructure is written in a programming language (infrastructure-as-code using TypeScript), and is powered AWS CloudFormation and AWS Cloud Development Kit (CDK).

Because of this, deploying the whole system (3 stacks) is as simple as installing the required tools (the AWS CDK cli and NodeJS), authenticating your AWS cli session (we use https://github.com/99designs/aws-vault [9]) and running `yarn cdk deploy --all`.

No other manual steps are required.

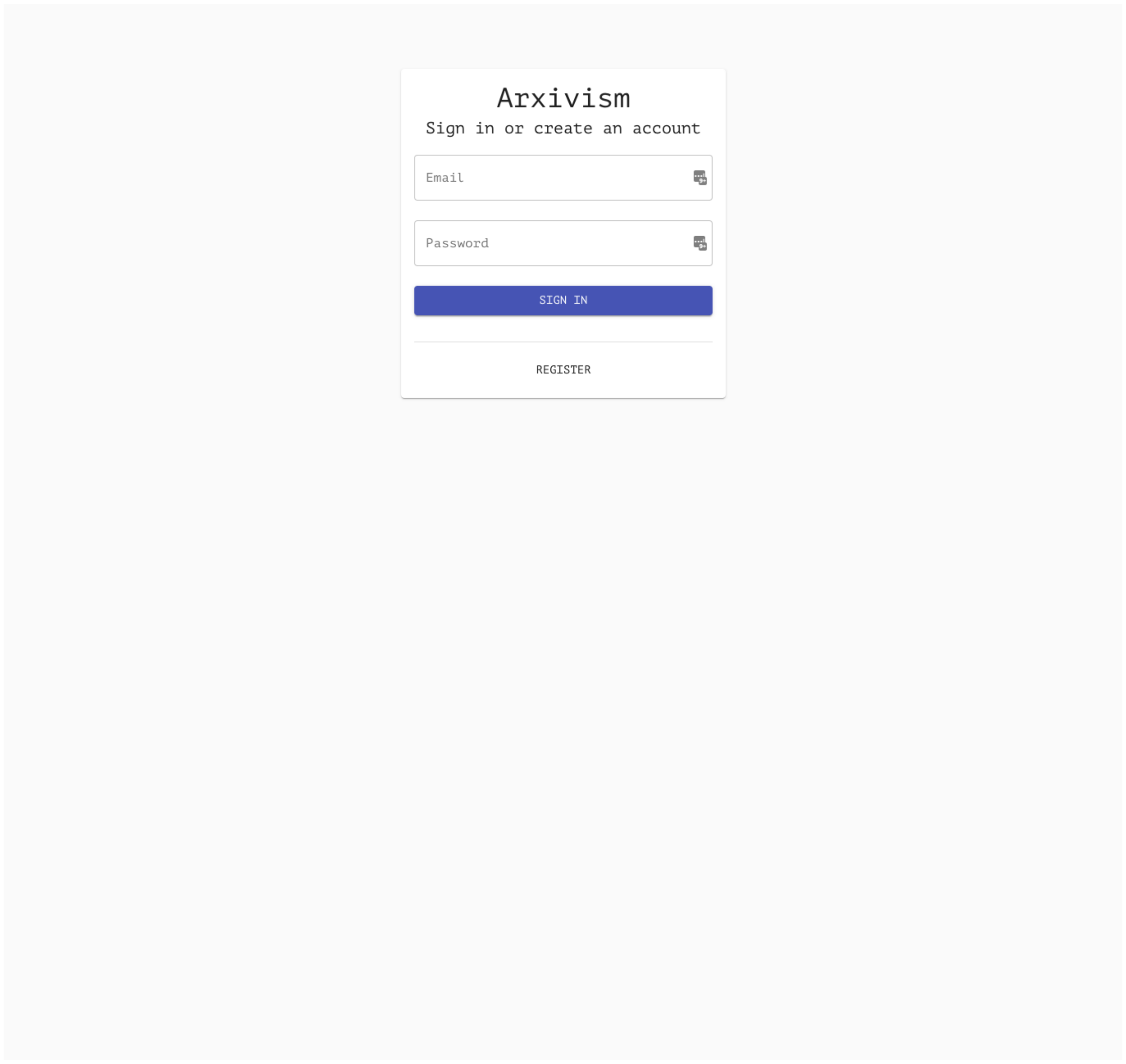
Steps

You don't have to follow these steps exactly (perhaps you already have NodeJS or know of more suitable installation means). This is just one potential way to install the required tools on your system.

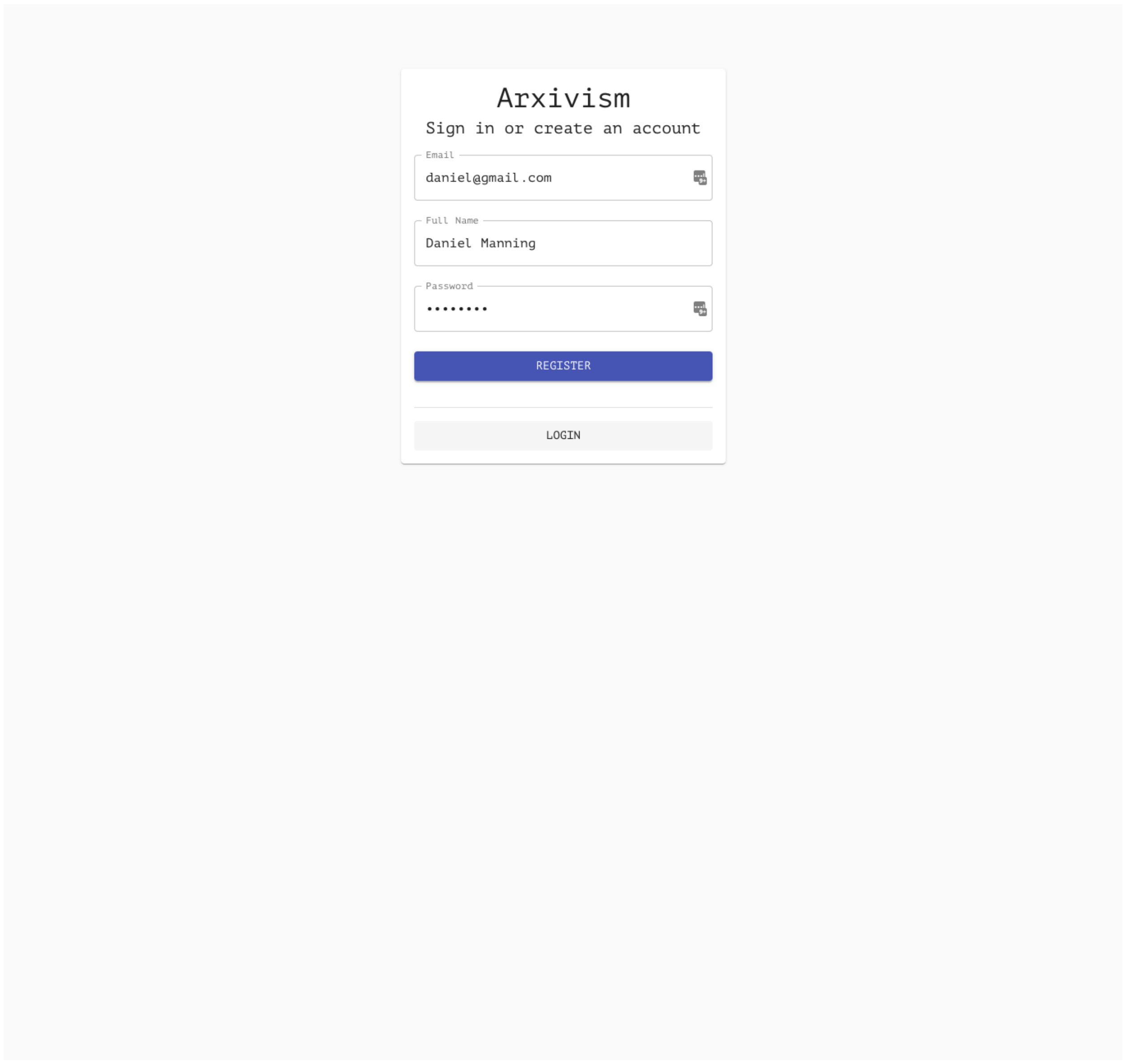
1. Install NodeJS on your system by using your system's specific download link from [2] https://nodejs.org/en/download/
2. Ensure you have the AWS CLI installed, configured, and authorized. A getting started guide can be found here [1] https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html
3. Install the AWS CDK Toolkit and follow the getting started guide [3] https://docs.aws.amazon.com/cdk/latest/guide/cli.html
4. Ensure you've run `cdk bootstrap` on your current AWS account so that the required CDK toolkit stack is initialised.
5. Clone the app's repository: https://github.com/dmisdm/cloud-computing-assignment-3
6. Install dependencies by running `yarn`
    1. If you don't have `yarn` installed you can install it by running `npm i -g yarn`
7. In the `/ops` directory, run `yarn cdk deploy --all` and approve any changes that are requested.
8. Once completed (it can take a while), you should be able to observe the Elastic Beanstalk frontend URL in the AWS web console, and use it to navigate to the application.

User Manual

Upon first arrival, you'll be asked to login or register if you haven't already. Proceed by doing one or the other.



Register for an account by entering each field and pressing the register button. You'll be logged in using that user automatically.


Arxivism

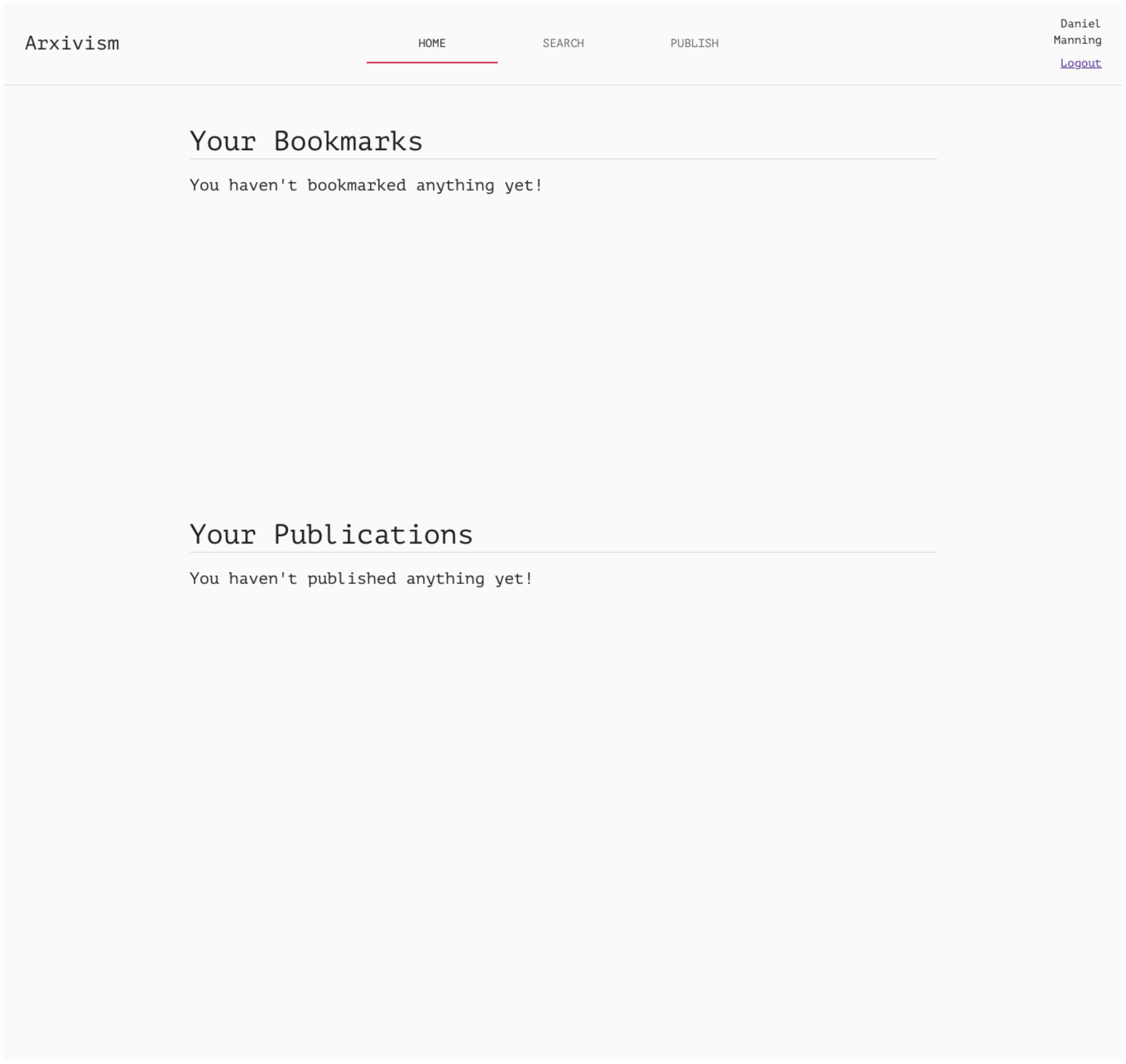
Sign in or create an account

Email: daniel@gmail.com

Full Name: Daniel Manning

Password: ••••••••

Your home page won't show any bookmarks or publications initially. Search for an article or navigate to the publish page to do either one.



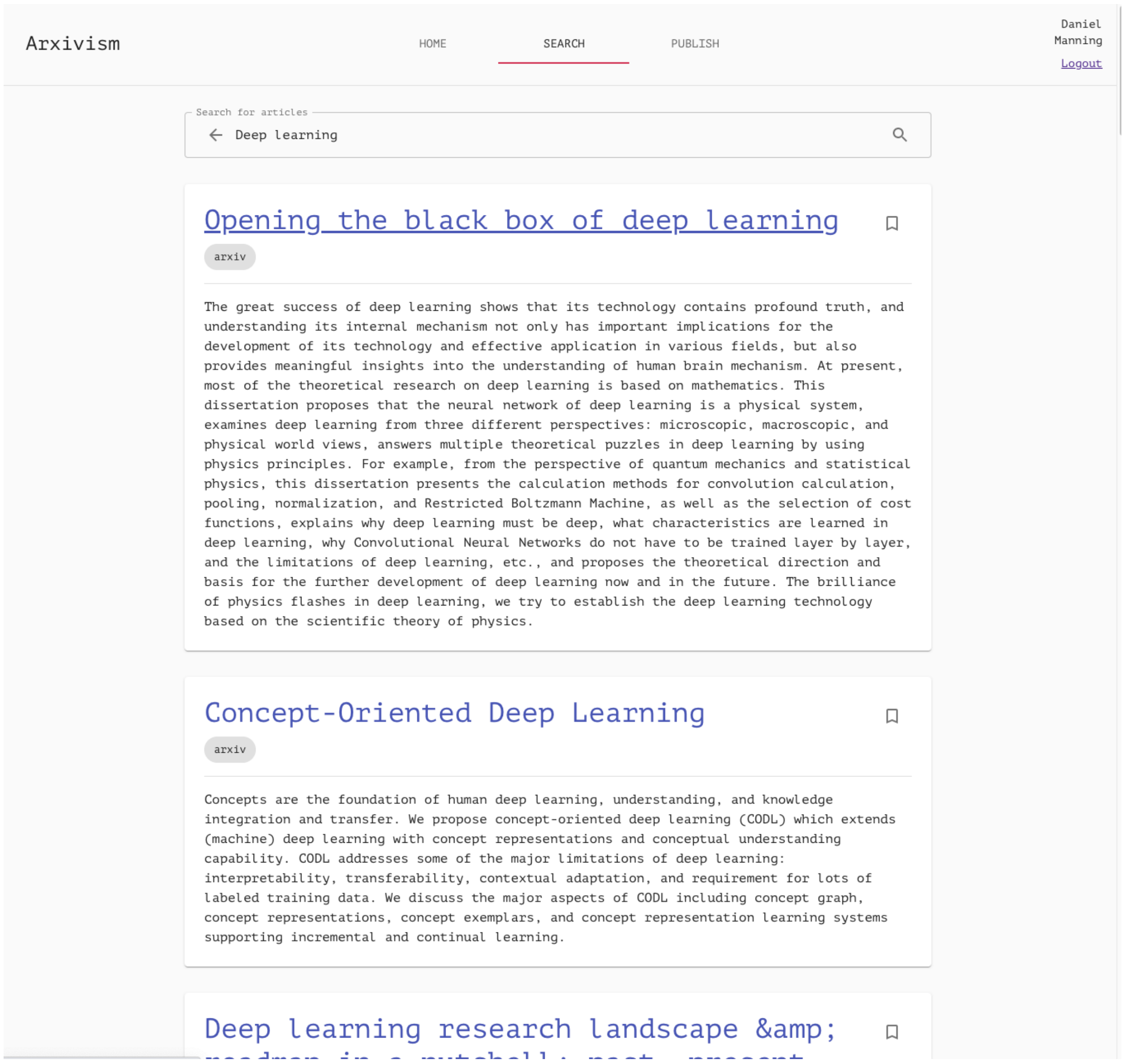
Your Bookmarks

You haven't bookmarked anything yet!

Your Publications

You haven't published anything yet!

To search for an article you can navigate to the search page and enter your query. In the results you'll have the ability to click on articles or bookmark them by clicking the icon.



Opening the black box of deep learning

arxiv

The great success of deep learning shows that its technology contains profound truth, and understanding its internal mechanism not only has important implications for the development of its technology and effective application in various fields, but also provides meaningful insights into the understanding of human brain mechanism. At present, most of the theoretical research on deep learning is based on mathematics. This dissertation proposes that the neural network of deep learning is a physical system, examines deep learning from three different perspectives: microscopic, macroscopic, and physical world views, answers multiple theoretical puzzles in deep learning by using physics principles. For example, from the perspective of quantum mechanics and statistical physics, this dissertation presents the calculation methods for convolution calculation, pooling, normalization, and Restricted Boltzmann Machine, as well as the selection of cost functions, explains why deep learning must be deep, what characteristics are learned in deep learning, why Convolutional Neural Networks do not have to be trained layer by layer, and the limitations of deep learning, etc., and proposes the theoretical direction and basis for the further development of deep learning now and in the future. The brilliance of physics flashes in deep learning, we try to establish the deep learning technology based on the scientific theory of physics.

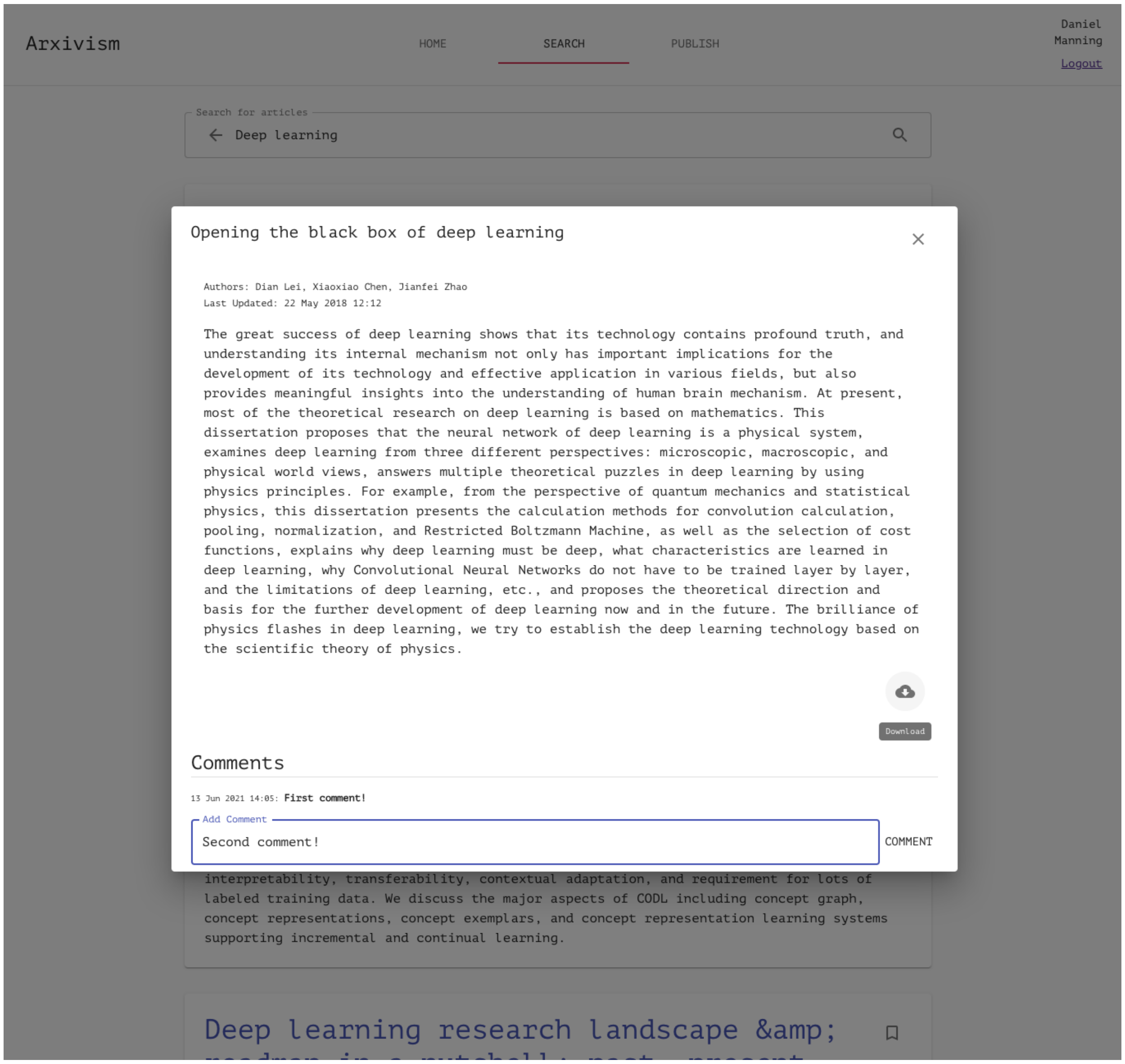
Concept-Oriented Deep Learning

arxiv

Concepts are the foundation of human deep learning, understanding, and knowledge integration and transfer. We propose concept-oriented deep learning (CODL) which extends (machine) deep learning with concept representations and conceptual understanding capability. CODL addresses some of the major limitations of deep learning: interpretability, transferability, contextual adaptation, and requirement for lots of labeled training data. We discuss the major aspects of CODL including concept graph, concept representations, concept exemplars, and concept representation learning systems supporting incremental and continual learning.

Deep learning research landscape &amp;

Clicking on an article shows you more information, and allows you to comment on it, or download the article.



Opening the black box of deep learning

Authors: Dian Lei, Xiaoxiao Chen, Jianfei Zhao
Last Updated: 22 May 2018 12:12

The great success of deep learning shows that its technology contains profound truth, and understanding its internal mechanism not only has important implications for the development of its technology and effective application in various fields, but also provides meaningful insights into the understanding of human brain mechanism. At present, most of the theoretical research on deep learning is based on mathematics. This dissertation proposes that the neural network of deep learning is a physical system, examines deep learning from three different perspectives: microscopic, macroscopic, and physical world views, answers multiple theoretical puzzles in deep learning by using physics principles. For example, from the perspective of quantum mechanics and statistical physics, this dissertation presents the calculation methods for convolution calculation, pooling, normalization, and Restricted Boltzmann Machine, as well as the selection of cost functions, explains why deep learning must be deep, what characteristics are learned in deep learning, why Convolutional Neural Networks do not have to be trained layer by layer, and the limitations of deep learning, etc., and proposes the theoretical direction and basis for the further development of deep learning now and in the future. The brilliance of physics flashes in deep learning, we try to establish the deep learning technology based on the scientific theory of physics.

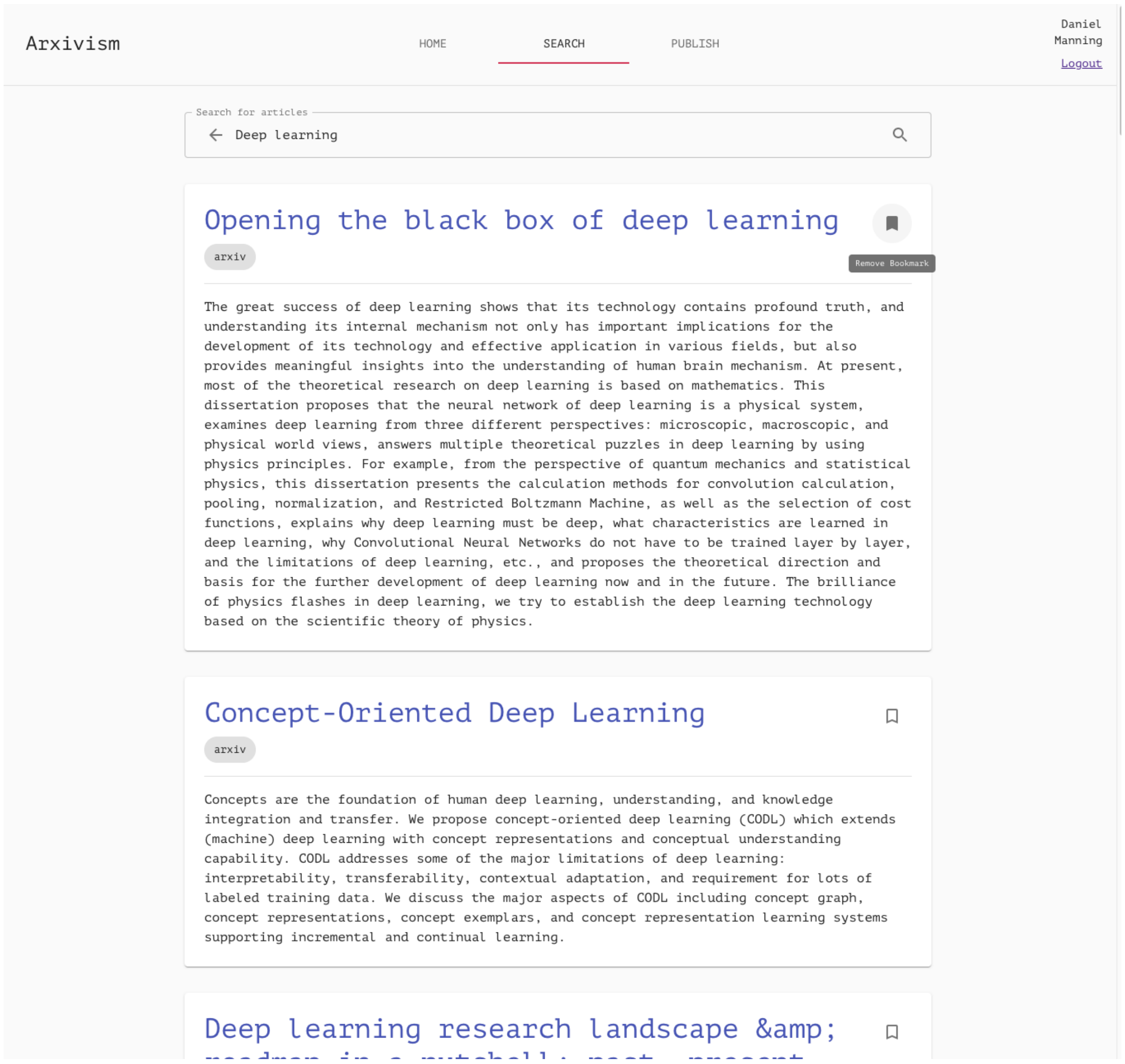
Comments

13 Jun 2021 14:05: **First comment!**

interpretability, transferability, contextual adaptation, and requirement for lots of labeled training data. We discuss the major aspects of CODL including concept graph, concept representations, concept exemplars, and concept representation learning systems supporting incremental and continual learning.

Deep learning research landscape &amp;

You may also remove bookmarks by clicking on the same icon.



Opening the black box of deep learning

arxiv

The great success of deep learning shows that its technology contains profound truth, and understanding its internal mechanism not only has important implications for the development of its technology and effective application in various fields, but also provides meaningful insights into the understanding of human brain mechanism. At present, most of the theoretical research on deep learning is based on mathematics. This dissertation proposes that the neural network of deep learning is a physical system, examines deep learning from three different perspectives: microscopic, macroscopic, and physical world views, answers multiple theoretical puzzles in deep learning by using physics principles. For example, from the perspective of quantum mechanics and statistical physics, this dissertation presents the calculation methods for convolution calculation, pooling, normalization, and Restricted Boltzmann Machine, as well as the selection of cost functions, explains why deep learning must be deep, what characteristics are learned in deep learning, why Convolutional Neural Networks do not have to be trained layer by layer, and the limitations of deep learning, etc., and proposes the theoretical direction and basis for the further development of deep learning now and in the future. The brilliance of physics flashes in deep learning, we try to establish the deep learning technology based on the scientific theory of physics.

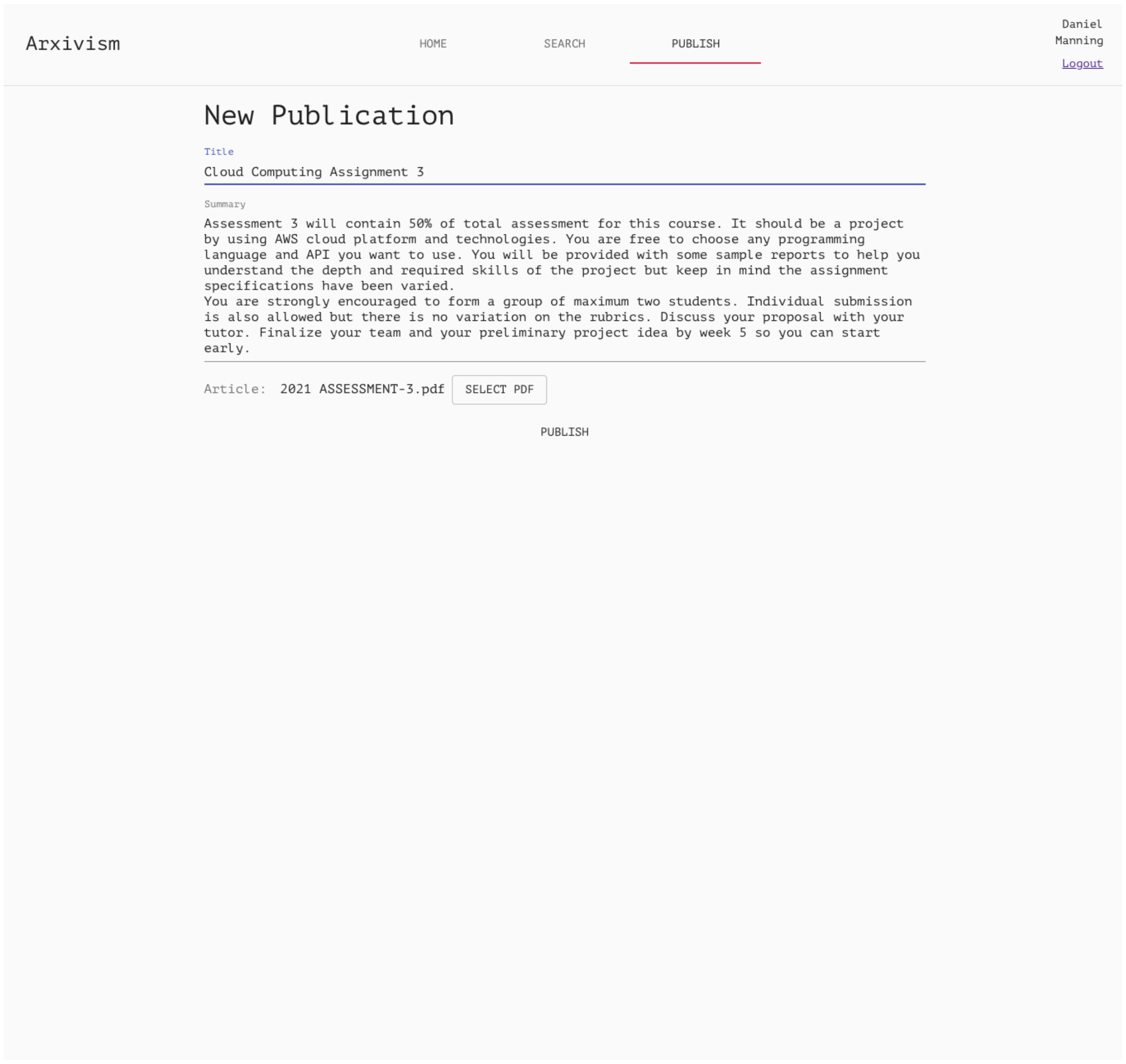
Concept-Oriented Deep Learning

arxiv

Concepts are the foundation of human deep learning, understanding, and knowledge integration and transfer. We propose concept-oriented deep learning (CODL) which extends (machine) deep learning with concept representations and conceptual understanding capability. CODL addresses some of the major limitations of deep learning: interpretability, transferability, contextual adaptation, and requirement for lots of labeled training data. We discuss the major aspects of CODL including concept graph, concept representations, concept exemplars, and concept representation learning systems supporting incremental and continual learning.

Deep learning research landscape &amp;

To publish an article, navigate to the publish page and enter the publication details along with a PDF to be uploaded, then press publish.



New Publication

Title

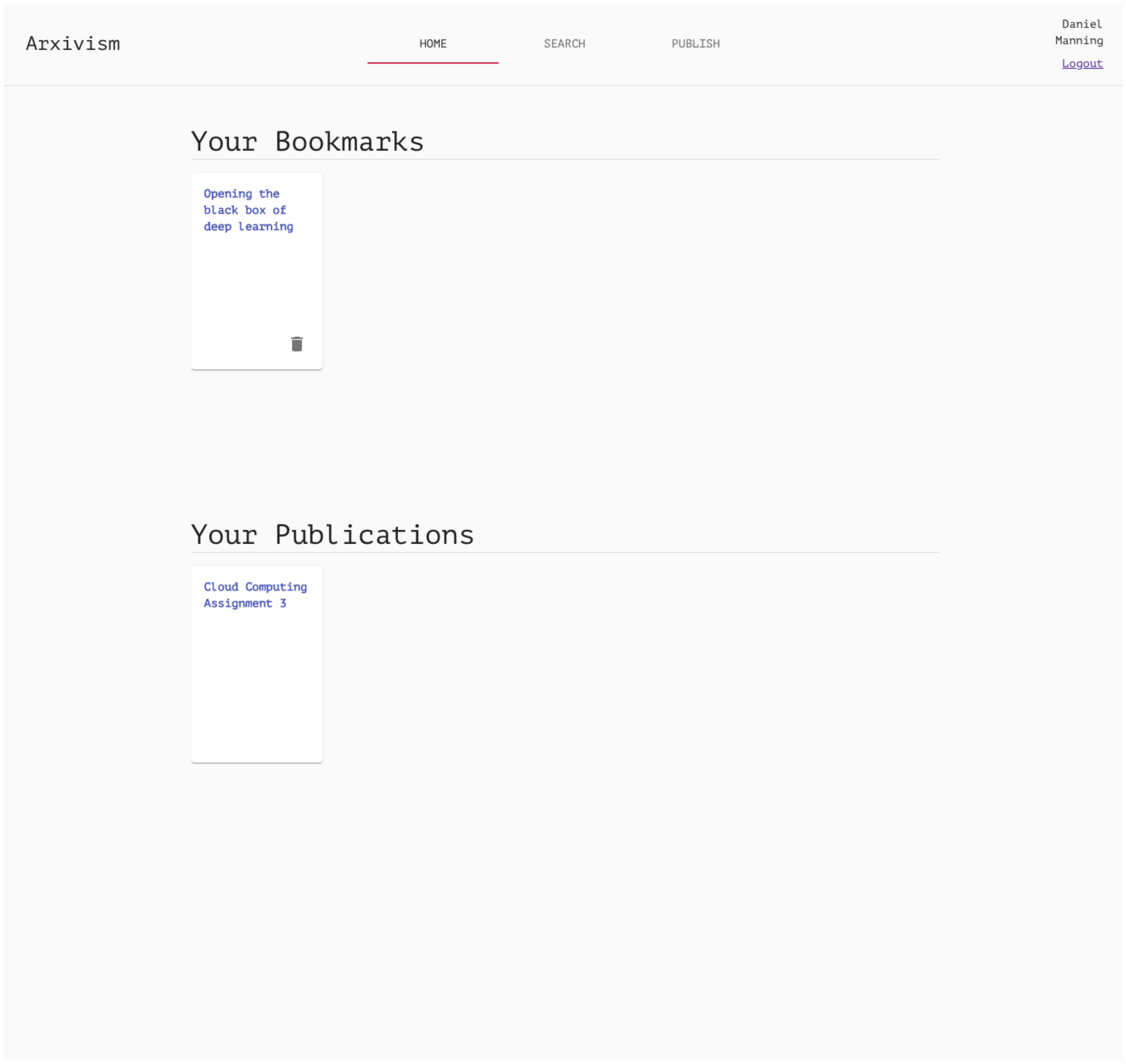
Cloud Computing Assignment 3

Summary

Assessment 3 will contain 50% of total assessment for this course. It should be a project by using AWS cloud platform and technologies. You are free to choose any programming language and API you want to use. You will be provided with some sample reports to help you understand the depth and required skills of the project but keep in mind the assignment specifications have been varied.
You are strongly encouraged to form a group of maximum two students. Individual submission is also allowed but there is no variation on the rubrics. Discuss your proposal with your tutor. Finalize your team and your preliminary project idea by week 5 so you can start early.

Article: 2021 ASSESSMENT-3.pdf

After bookmarking or publishing something, your home page will display them, allowing you to view their information or remove any bookmarks.



Your Bookmarks

Opening the black box of deep learning

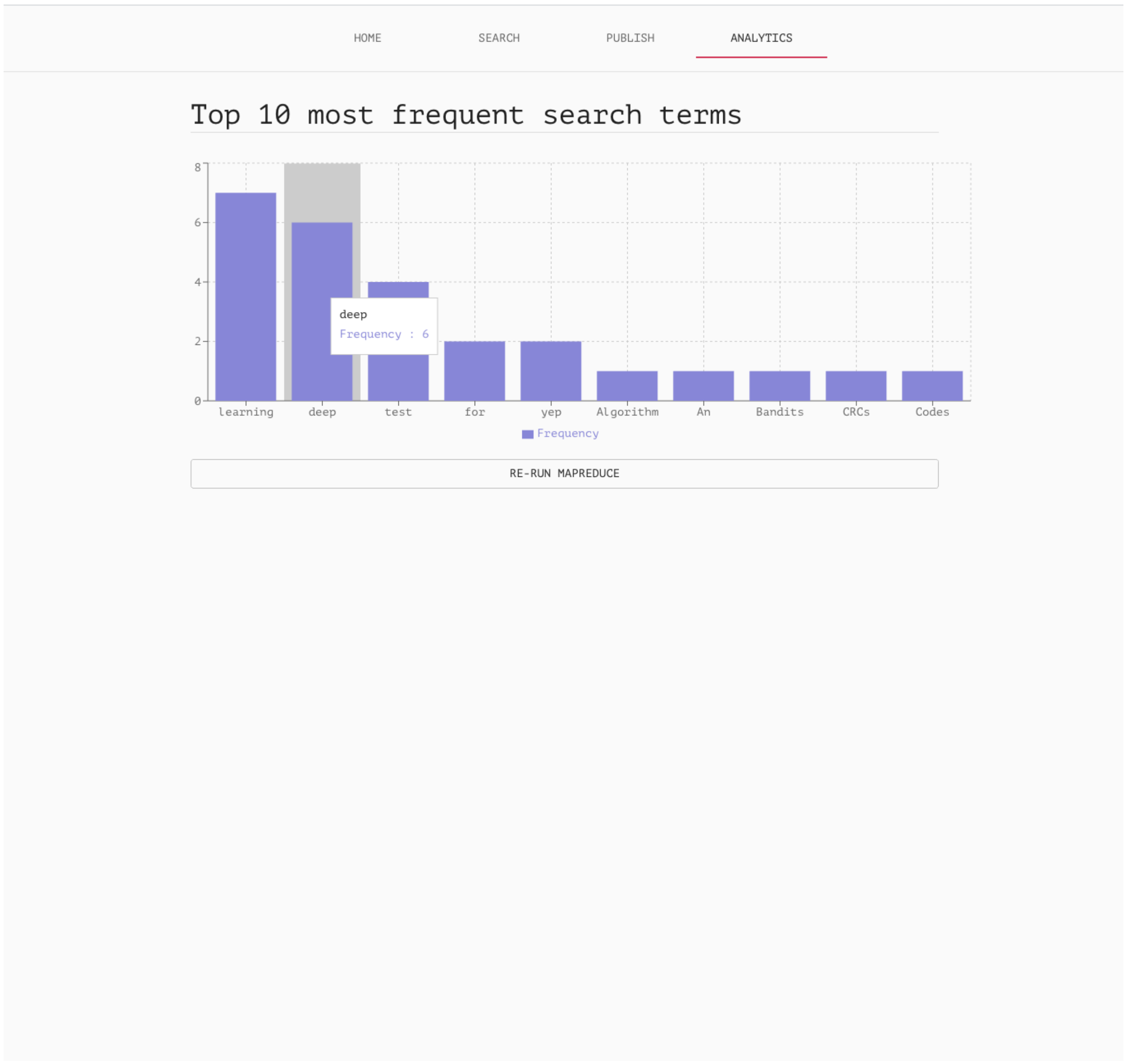
Your Publications

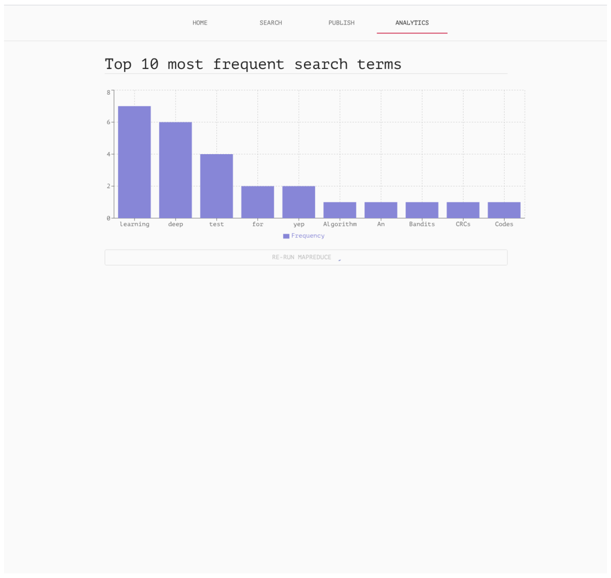
Cloud Computing Assignment 3

Admin Analytics

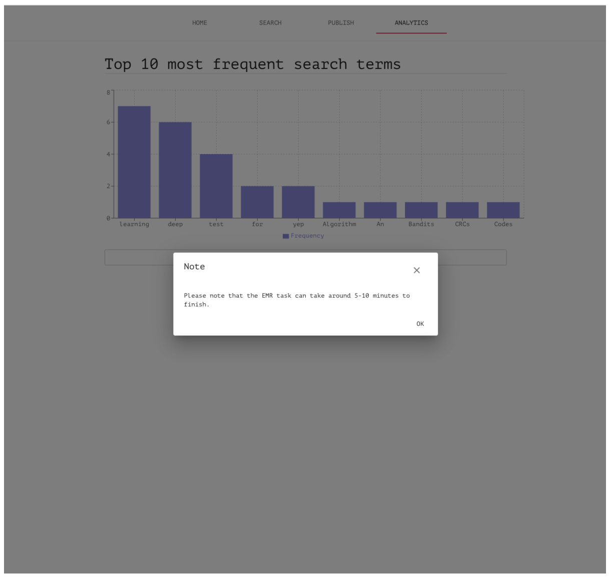
If you are an administrator, you'll be able to navigate to the analytics page.

Here we can view the top 10 most frequent search terms across all user queries, and update the results by re-running our MapReduce function.


HOME
SEARCH
PUBLISH
ANALYTICS
Top 10 most frequent search terms
8
6
4
2
0
learning
deep
test
for
yep
Algorithm
An
Bandits
CRCs
Codes
deep
Frequency : 6
Frequency
RE-RUN MAPREDUCE



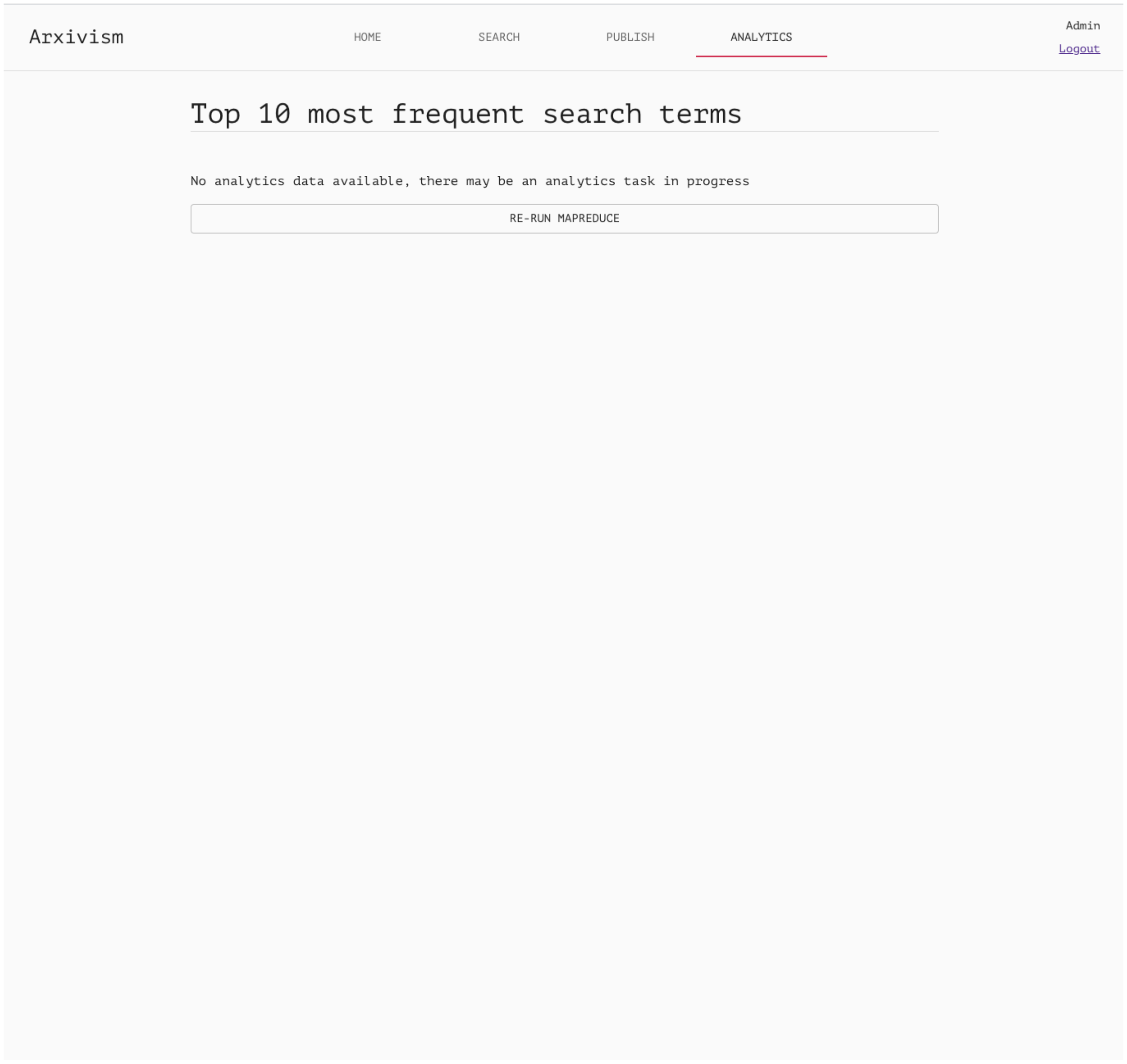
Top 10 most frequent search terms



Top 10 most frequent search terms

Note

Please note that the EMR task can take around 5-10 minutes to finish.



Top 10 most frequent search terms

No analytics data available, there may be an analytics task in progress

References

[1] "Configuration basics - AWS Command Line Interface," Amazon.com, 2021. https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html (accessed Jun. 13, 2021).

[2] Node.js, "Download | Node.js," Node.js, 2021. https://nodejs.org/en/download/ (accessed Jun. 13, 2021).

[3] “AWS CDK Toolkit (cdk command) - AWS Cloud Development Kit (CDK),” Amazon.com, 2021. https://docs.aws.amazon.com/cdk/latest/guide/cli.html (accessed Jun. 13, 2021).

[4] “arXiv.org e-Print archive,” Arxiv.org, 2021. https://arxiv.org/ (accessed Jun. 13, 2021).

[5] “arXiv Vanity – Read academic papers from arXiv as web pages,” Arxiv-vanity.com, 2021. https://www.arxiv-vanity.com/ (accessed Jun. 13, 2021).

[6] “Researcher | An App For Academics,” Researcher-app.com, 2020. https://www.researcher-app.com/ (accessed Jun. 13, 2021).

[7] “Home | Vite,” Vitejs.dev, 2019. https://vitejs.dev/ (accessed Jun. 13, 2021).

[8] “arXiv API Access | arXiv e-print repository,” Arxiv.org, 2021. https://arxiv.org/help/api/ (accessed Jun. 13, 2021).

[9] 99designs, “99designs/aws-vault,” GitHub, May 24, 2021. https://github.com/99designs/aws-vault (accessed Jun. 13, 2021).

[10] “Prisma - Next-generation Node.js and TypeScript ORM for Databases,” Prisma, 2018. https://www.prisma.io/ (accessed Jun. 13, 2021).