ILLUMINATE YOUR DATA

Shiny + Amazon Web Service Cloud Development Kit

Zuri Hunter
Data Engineer
Data and Analytics

Zuri Hunter

Data Engineer
Data Analytics

WHAT YOU WILL LEARN

- Shiny
- AWS CDK
- Docker Integration
- CDK Code Walkthrough
- Questions

NFL DATA & ANALYTICS

What is Shiny?

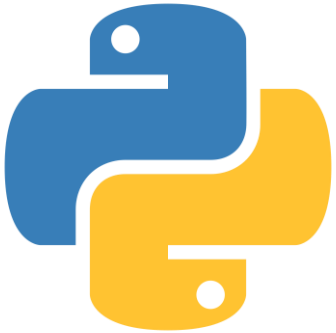
It is a **free open source** web framework for building web application in both **R** and **Python**.

Founded in 2012 and is one of the oldest web application frameworks for data visualization and analysis.

Seamless integration with both R and Python libraries.

Leverages **reactive programming** for the UI.

It's used a lot in the space of sports data visualization.



Shiny in Comparison

	Shiny	Dash	Streamlit	Bokeh
Language	**Python, R**	Python, Julia, R , F#	Python	Python
Backend Architecture	**Stateful**	Stateless	Stateful	Stateful
Web Protocol	**Websockets**	HTTP(S)	Websockets	Websockets
App Structure	**Single/Multi Page**	Multi Page	Multipage	Single Page
Styling Control	**Bootstrap/CSS**	Bootstrap/CSS	CSS	Custom

What is AWS CDK?

It is an **open source** software development framework for defining **cloud infrastructure in code** and provisioning it through **AWS CloudFormation**.

Two Primary Parts

- Construct Library
- CLI

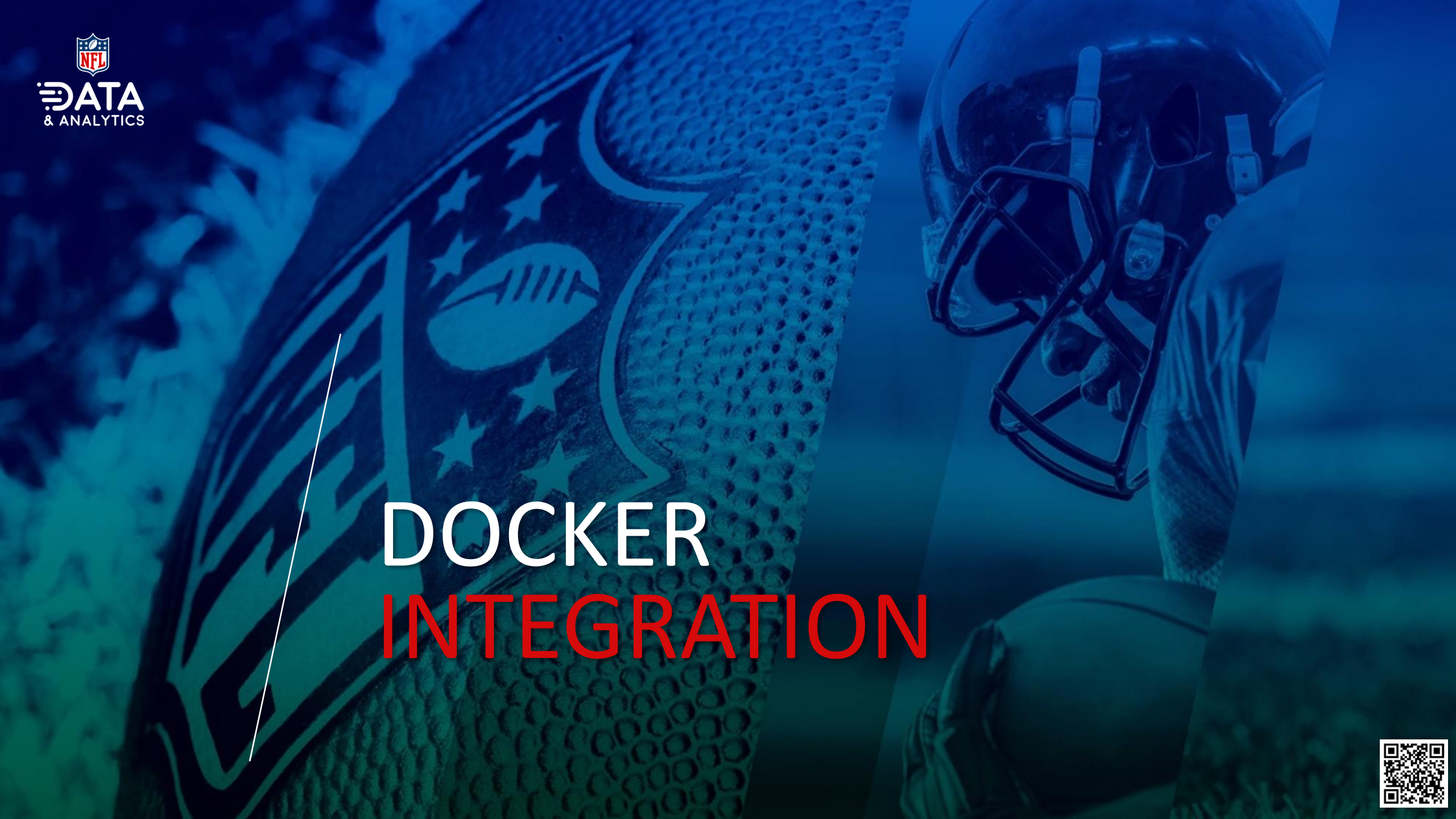
Supported languages are **TypeScript, JavaScript, Python, Java, C#/.NET** and **Go.**

Best Practices

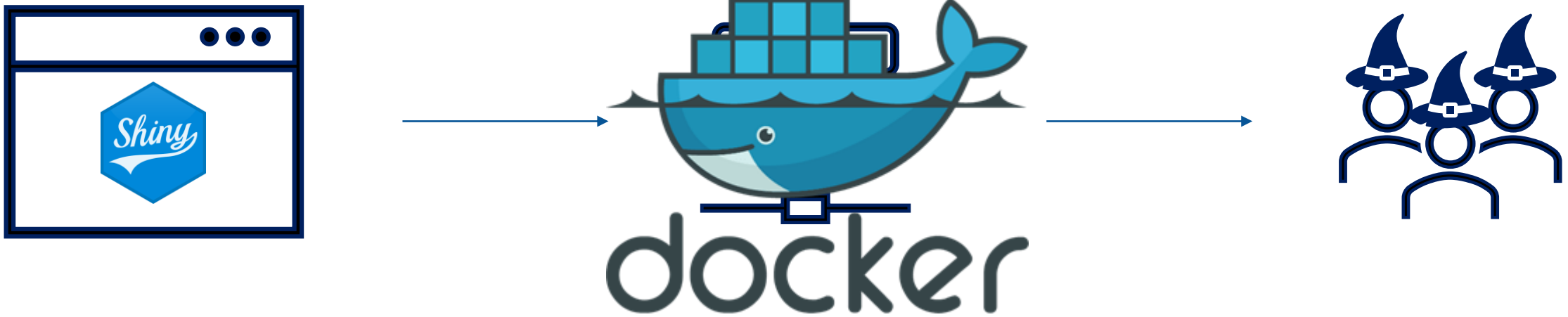
- **Don’t** use *security defaults* when it comes to services roles and permissions.
- **Don’t** store *secrets* in your stacks.
- **Organize** your AWS resources and stacks in a way where that aren’t *too dependent* on each other.
- **Build** through *AWS Console* first and then write the infrastructure code.

Benefits of AWS CDK

- Easily **convert** the code into a Cloudformation template.
- **Records** the state of the architecture upon each deployment.
- Allows the ability to create **reuseable** code and distribute amongst others.
- Easy **collaboration** between Data Scientist and DevOps Engineers.
- **Enforces** security and best practices when it comes to deploying resources to AWS.

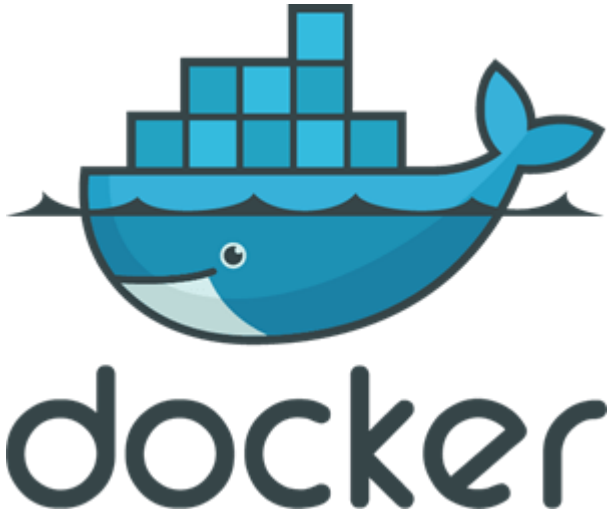

NFL
DATA
& ANALYTICS
DOCKER
INTEGRATION

http://localhost:3000



Docker

Is a software platform that empowers everyone to build, share and run applications on any platform using containers.



Shiny Dockerfile

```
FROM rocker/shiny
RUN apt-get update && apt-get install -y python3-pip
RUN . /etc/environment && R -e "install.packages(c('ROCR', 'gbm'),
repos='$MRAN')" \
USER shiny
WORKDIR /srv/shiny-server
COPY shiny/requirements.txt /srv/shiny-server/
RUN pip install --no-cache-dir -r requirements.txt
COPY shiny /srv/shiny-server
EXPOSE 8000
RUN ls -la /srv/shiny-server

CMD ["shiny", "run", "--host", "0.0.0.0", "--port", "8000", "main.py"]
```

- This file serve as instructions on how to install and launch your application.
- Run **`docker build [Path/To/Dockerfile] -t [Name_Of_Image]`**
- Run **`docker run [Name_Of_Image] -p 8000:8000`**
- Go to **`http://localhost:8000`**

NFL DATA & ANALYTICS

CDK CODE WALKTHROUGH

Getting Started

CDK Preparation

1. Have an empty directory for the project.
2. Run `cdk init app --language=python` in the empty directory.

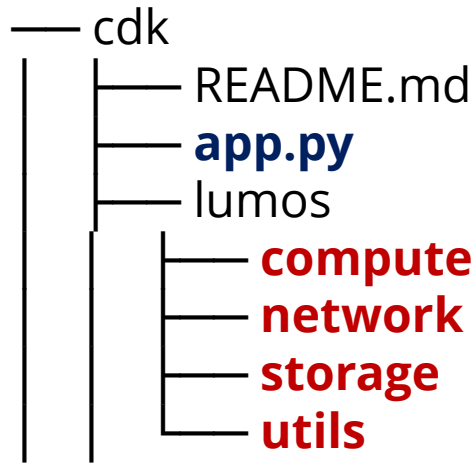
AWS Preparation

1. Create a **user** that will have access to create, read, update and delete the following resources:
    1. Elastic Container Registry
    2. Elastic Container Service
    3. S3
    4. IAM Role
    5. IAM Policy
    6. SSM

2. Set that new user as your AWS Profile with its **Access Key ID** and **Secret Access Key**.
3. Purchase **domain** name from Route53.

Deployment Workflow + File Structure

1. Elastic Container Registry (Storage)**
2. Simple Storage Solutions- S3 (Storage)**
3. Security Group (Network)
4. Application Load Balancer (Network)
5. Route 53 (Network)**
6. Elastic Container Service (Compute)



ECR Stack

- Defines the registry in ECR.
- Anytime we build and push the docker image it is placed in the registry.
- Only run this once.
- Run **`cdk deploy EcrStack`** to launch the stack.

```python
class EcrStack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # # # Create WISD24 Lumos ECR Repository
        self.repository = ecr.Repository(
            self, "LumosRepository",
            repository_name="wisd24/lumos-shiny-application",
            removal_policy=RemovalPolicy.DESTROY
        )
```

S3 Stack

```python
class S3Stack(Stack):

    def __init__(self, scope: Construct, construct_id: str, **kwargs) -> None:
        super().__init__(scope, construct_id, **kwargs)

        # # # Create WISD24 Lumos Data S3 Bucket
        self.bucket = s3.Bucket(
            self, "LumosDataBucket",
            bucket_name="wisd24-lumos-data-bucket",
            removal_policy=RemovalPolicy.DESTROY,
            auto_delete_objects=True
        )

        # # # Zip the data directory for upload to S3
        base_dir = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
        data_dir = os.path.join(base_dir, '../data/')

        zip_dir('/Users/zuri/Documents/Explore/lumos/data', 'data.zip')

        s3deploy.BucketDeployment(
                self, "LumosDeployData",
                sources=[s3deploy.Source.asset(data_dir + "../../data/data.zip")],
                destination_bucket=self.bucket,
                retain_on_delete=False
        )
```

- Define the name of the name of the bucket.
- Compressed the *.csv files that the application will read.
- Add utility to deploy the bucket with the *.csv files.
- Run **`cdk deploy S3Stack`** to deploy the stack.

Security Group Stack

- Define the Virtual Private Cloud (VPC) to place the Security Group.

```python
class SecurityGroupStack(Stack):
    def __init__(self, scope: Construct, id: str, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        # # # Create WISD24 Lumos VPC
        self.vpc = ec2.Vpc(
            self,
            'LumosVPC',
            max_azs=2,
            subnet_configuration=[
                ec2.SubnetConfiguration(
                    name='Public',
                    subnet_type=ec2.SubnetType.PUBLIC,
                    cidr_mask=24
                ),
                ec2.SubnetConfiguration(
                    name='Private',
                    subnet_type=ec2.SubnetType.PRIVATE_WITH_NAT,
                    cidr_mask=24
                )
            ],
            nat_gateways=1
        )
```

Security Group Stack (cont.)

```
#### SecurityGroupStack Continued....

 # # # Create WISD24 Lumos Security Group
     self.security_group = ec2.SecurityGroup(
         self, "LumosSecurityGroup",
         vpc=self.vpc,
         description="Allow inbound traffic on port 80",
         allow_all_outbound=True
     )

 # # # Add Ingress Rules for traffic to come in through port 80 and port 8000, respectively
     self.security_group.add_ingress_rule(
         peer=ec2.Peer.any_ipv4(),
         connection=ec2.Port.tcp(80),
         description="Allow inbound traffic on port 80"
     )

     self.security_group.add_ingress_rule(
         peer=ec2.Peer.any_ipv4(),
         connection=ec2.Port.tcp(8000),
         description="Allow inbound traffic on port 80"
     )
```

- Create the Security and attach the newly created VPC.
- Add permissions for internet traffic to flow to port 80 and 8000 within the group.
- Run **`cdk deploy SecurityGroup`** to launch the Security Group and VPC.

Application Load Balancer Stack

- Define the load balancer to be an Application Load Balancer.
- Add a listener to the load balancer for traffic to flow on port 80.
- Run **`cdk deploy LoadBalancerStack`** to launch the Load Balancer and Listener.

```python
class LoadBalancerStack(Stack):
    @property
    def load_balancer(self):
        return self._load_balancer
    def __init__(
            self,
            scope: Construct,
            id: str,
            vpc: ec2.IVpc,
            security_group: ec2.ISecurityGroup,
            **kwargs) -> None:
        super().__init__(scope, id, **kwargs)
        self._load_balancer = elbv2.ApplicationLoadBalancer(
            self,
            'LumosLoadBalancer',
            vpc=vpc,
            internet_facing=True
        )
        self._listener = self.load_balancer.add_listener(
            "LumosListener",
            port=80,
            open=True
        )
```

Route53 Stack

```python
class DNSStack(Stack):
    def __init__(
            self,
            scope: Construct,
            id: str,
            load_balancer,
            **kwargs) -> None:
        super().__init__(scope, id, **kwargs)

        hosted_zone_id = 'Z00528603RL8W30CQULIV'

        hosted_zone = route53.HostedZone.from_hosted_zone_attributes(
            self,
            "LumosHostedZone",
            hosted_zone_id=hosted_zone_id,
            zone_name='illuminatewithlumos.com'
        )

        route53.ARecord(
            self,
            'LumosAliasRecord',
            record_name='www',
            target=route53.RecordTarget.from_alias(targets.LoadBalancerTarget(load_balancer.load_balancer)),
            zone=hosted_zone,
        )
```

- Hosted Zone ID is from the Domain Name purchased in Route 53.
- **Hosted Zone** is a container with information on how to route traffic to Domain names.
- Create an **A Record** that will say the domain relates to the URL of our Application Load Balancer.
- Only run this once.
- Run `cdk deploy DNSStack` to deploy those resources.

Elastic Container Service Stack

- Log Group is required for launching a container within the cluster.
- Define the cluster and connect it to the existing VPC.
- Define the Task Definition which is defining the blueprint of the container that is going to host the application.

```python
class FargateStack(Stack):
    def __init__(
            self,
            scope: Construct,
            id: str,
            ecr_repository: ecr.IRepository,
            vpc: ec2.IVpc,
            load_balancer,
            s3_bucket: s3.IBucket, **kwargs) -> None:
        super().__init__(scope, id, **kwargs)


        log_group = logs.LogGroup(
            self,
            'LumosServiceLogGroup',
            log_group_name='/aws/ecs/lumos-service',
            retention= logs.RetentionDays.FIVE_DAYS,
            removal_policy= RemovalPolicy.DESTROY
        )

        cluster = ecs.Cluster(
            self,
            "LumosCluster",
            vpc=vpc
        )

        task_definition = ecs.FargateTaskDefinition(
            self,
            "LumosTaskDefinition",
            cpu=256,
            memory_limit_mib=512
        )
```

Elastic Container Service Stack (cont.)

```
### Fargate Service Cont...
        lumos_docker_image = ecs.ContainerImage.from_ecr_repository(
            repository=ecr_repository,
            tag="latest"
        )

        container = task_definition.add_container(
            "LumosContainer",
            image=lumos_docker_image,
            memory_reservation_mib=512,
            logging= ecs.AwsLogDriver(
                log_group=log_group,
                stream_prefix='lumos-service'
            )
        )

        container.add_port_mappings(
            ecs.PortMapping(
                container_port=8000
            )
        )
```

- Connect the Docker image that is in ECR and add it as a container.
- Expose port 8000 within the container for the Shiny application to be reached.

Elastic Container Service Stack (cont.)

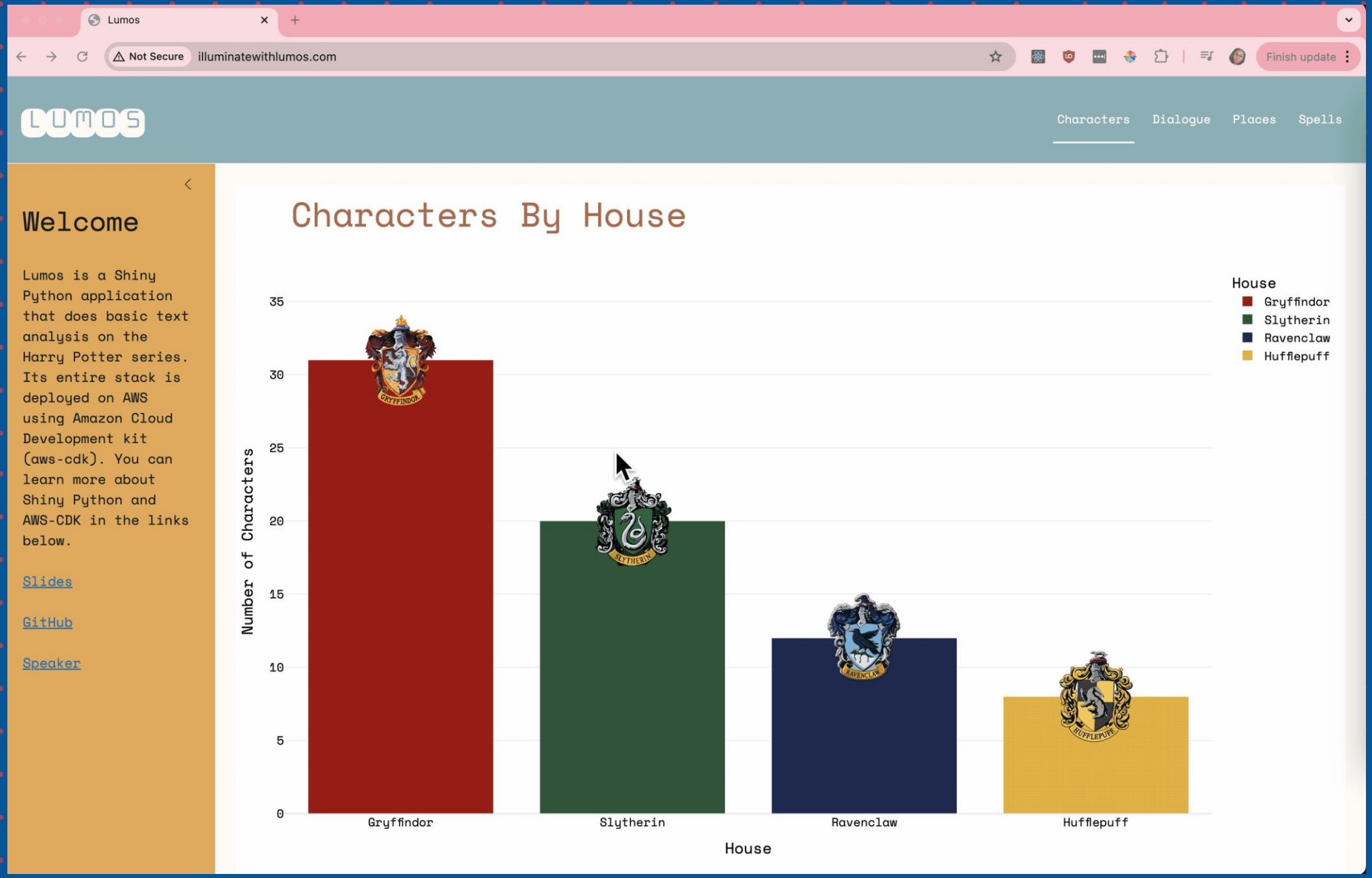
- Define the service within the cluster and say that it should always have two containers.
- Grant the cluster access to read the bucket.
- Connect the listener to direct traffic to our container.
- Run **`cdk deploy FargateService`** to launch the entire compute resource.

```
### Fargate Service Cont...
      fargate_service = ecs.FargateService(
          self,
          "LumosFargateService",
          cluster=cluster,
          task_definition=task_definition,
          desired_count=2
      )

      s3_bucket.grant_read(task_definition.task_role)

      load_balancer._listener.add_targets(
          'LumosTargetGroup',
          port=80,
          targets=[fargate_service]
      )
```

GIF DEMO


Lumos
Not Secure
illuminatewithlumos.com
Finish update
LUMOS
Characters
Dialogue
Places
Spells
Welcome
Lumos is a Shiny Python application that does basic text analysis on the Harry Potter series. Its entire stack is deployed on AWS using Amazon Cloud Development kit (aws-cdk). You can learn more about Shiny Python and AWS-CDK in the links below.
Slides
GitHub
Speaker
Characters By House
House
Gryffindor
Slytherin
Ravenclaw
Hufflepuff
Number of Characters
0
5
10
15
20
25
30
35
Gryffindor
Slytherin
Ravenclaw
Hufflepuff
House

SUMMARY

- Shiny is great web framework to share data visualization supports both R and Python.
- Docker is reliable tool to use for building, running and shipping application to any platform.
- AWS CDK provides seamless collaboration between Data Scientist and DevOps Engineers for deploying Shiny applications.
- Deploying Shiny applications through CDK helps enforce best security practices and track costs amongst AWS resources.

QUESTIONS

Zuri Hunter
Data Engineer
Data and Analytics

RESOURCES

- Outstanding User Interfaces with Shiny - https://unleash-shiny.rinterface.com/
- Advanced AWS CDK: Lessons Learned from 4 Years of Use - https://www.youtube.com/watch?v=Wzawix9bMAE
- AWS Cloud Development Kit Crash Course - https://www.youtube.com/watch?v=T-H4nJQyMig
- Shiny Proxy - https://www.shinyproxy.io/
- Docker for Beginners - https://docker-curriculum.com/
- CDK Nag - https://github.com/cdklabs/cdk-nag

DATA & ANALYTICS