Requirements

- You will need a domain name (one has been created for you for this workshop).
- You need git for the various steps of deployment. If you are working on a Mac, you already have git installed. If you are working on a PC, make sure you install git and familiarize yourself with the basic commands before proceeding.
- You will need to install npm.
- You will need to install Docker and create an account.
- You will need a Postgres manager. For example, you can download SQLPro for Postgres.

Important Note: Make sure you replace all the relevant names in any commands you execute. Names that need to be replaced will be highlighted and will follow this format: NAME_OF_FILE.

Getting Started

Needed:

- Pushkin
- Pushkin-cli
- short-quiz

The version of Pushkin that will be used for the workshop can be found here. Use the following command to clone it:

```
git clone [URL]
```

You will also need to clone the **pushkin-cli tool**, which can be found here. Both repositories should be cloned in the same directory.

Once you have a copy of Pushkin, and you can add the test experiment that you will be using for deployment. A short quiz has been created for this tutorial. There are a number of files for the back end and the front end of the quiz that need to be placed in the appropriate directories.

All the files for the workshop's test experiment can be found here. Clone the **short_quiz** repository. Inside, you will find 2 folders: **front-end** and **back-end**, and a few supplementary files.

All the files contained in the **back-end** folder can be placed inside a new folder named **short-quiz** inside the **experiments** folder in the **pushkin_workshop** folder.

The files in the **front-end** folder will need to be placed inside the **pushkin_workshop** -> **front-end** -> **experiments** folder in a new folder named **short-quiz**.

Additionally, you will need to add the route for the quiz. The file containing all routes (**routes.js**) can be found in **pushkin_workshop** -> **front-end** -> **core**. At the top, add an import statement for the quiz component:

```
import ShortQuiz from '../experiments/short-quiz/index';
```

and a route below line 67 in the following format:

```
  <Route
      path="/quizzes/short-quiz"
      component={ShortQuiz}
      onEnter={ensureDesktop}
    />
```

You will need to add your new experiment on the **Quizzes** page. Go to **pushkin-workshop** -> **front-end** -> **pages** -> **quizzes** and open **index.js**. Add the quiz below line 36 in the following format:

```
 <p className={s.mb25}>
                    <Link className={s.title} to="/quizzes/short-quiz">
                      The Short Quiz
                    </Link>
                    <br /> <br />This is essentially the Vocab Quiz, but without all
the demographics.
                  </p>
```

Finally, you will need to add your experiment to **docker-compose.debug.yml** (located in the **pushkin_workshop** folder). Add the quiz as the last portion of the file in the following format:

```
short-quiz:
    image: 'pushkin/short-quiz:latest'
    build:
      context: ./workers/short-quiz
      dockerfile: Dockerfile
    volumes:
      - './workers/short-quiz:/usr/src/app'
    command: bash start.debug.sh
    depends_on:
      - message-queue
    environment:
      - 'AMPQ_ADDRESS=amqp://message-queue:5672'
      - QUEUE=short-quiz
    links:
      - message-queue
```

Your test experiment should be ready for deployment!

First Steps for Local Deployment

Setting Up Databases

When testing locally, you will use a local Postgres database. If you want to have easy access to the database tables to see stimuli and responses, you should set up your Postgres manager to have access to the local Postgres databases.

Open your Postgres manager, select **New** and fill out the following fields to establish a connection to the main database:

Server host: localhost
Login: postgres
Server port: 5432
Database: dev

Establish a connection to the transactions database with the following information:

Server host: localhost
Login: postgres
Server port: 5433
Database: transactions_dev

Pushkin-cli

The pushkin tool is created to easily scaffold or delete a set of routes, migrations, bookshelf models, seeds and workers for a new quiz. To use the **pushkin-cli** tool you will need to install the dependencies in the current working directory as a global package. Here is how to do it:

- Open Terminal
- Change your working directory to the folder containing the cloned **pushkin-cli** repository and install:

```
$ cd pushkin-cli

$ npm install –g ./
```

Note: If you are having issues with permissions, run

```
sudo npm install -g ./
```

To test whether the **pushkin-cli** installation was successful, run the command `pushkin`. If the installation was successful, you will get a list of commands that you can execute:

```
Usage: pushkin [options] [command]

Options:

  -h, --help  output usage information

Commands:

  generate [thing] [name]  generate a new controller, model, or worker
  sync                     sync experiment files
  delete [thing] [name]    delete a controller
  list [thing]             show list of entities
  scaffold [name]          Generate a brand new quiz
  seed [name]              please run node seeder.js [name] in pushkin-db docker
  help [cmd]               display help for [cmd]
```

- Change the working directory back to **pushkin_workshop** (`cd pushkin_workshop`)
- run `pushkin sync`

Pushkin sync is a command which ensures that all back end experiment files (workers, seeds, migrations, models) are distributed to their right location for Pushkin to use them.

Important Note: You should store all the back end experiment files in the **experiments** folder located inside the **pushkin_workshop** folder. The **experiments** folder should contain a folder for each quiz you would like to deploy, and this folder should contain only the back end files for the quiz. Running `pushkin sync` will copy and move those files to their respective locations to be used by Pushkin.

Setting Up a Running Docker Container

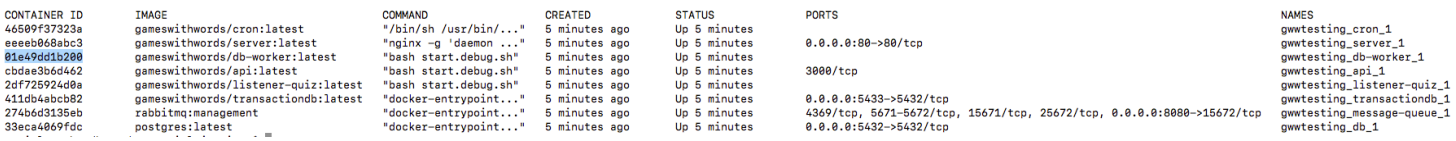
Docker provides consistency across environments from development to production and allows you to deploy your application in containers that can be started and stopped at your convenience. Provided that you have installed Docker, you should be able to start a Docker container with your Pushkin application by running the following command in the **pushkin_workshop** directory:

```
$ docker-compose -f docker-compose.debug.yml up
```

Compose is a tool for defining and running multi-container Docker applications. With Compose, you use a YAML file to configure your application's services. Then, with a single command, you create and start all the services from your configuration.

Starting Docker containers can take a couple of minutes. If you are not sure whether all the containers are running yet, you can run `docker ps` from your Terminal to check (`docker ps` gives a list of all running containers).

When all the containers are up and running, run `docker ps` from any directory. You will get a list of running containers with their IDs, STATUS, PORTS, and NAMES. You will need to get a bash shell in the **db-worker** container in order to run your migrations and seed the stimuli database table with the stimuli for the quiz. To do that, copy the container ID of `db-worker` ,



CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
46509f37323a	gameswithwords/cron:latest	"/bin/sh /usr/bin/..."	5 minutes ago	Up 5 minutes		gwwtesting_cron_1
eeeeb068abc3	gameswithwords/server:latest	"nginx -g 'daemon ..."	5 minutes ago	Up 5 minutes	0.0.0.0:80->80/tcp	gwwtesting_server_1
01e49dd1b200	gameswithwords/db-worker:latest	"bash start.debug.sh"	5 minutes ago	Up 5 minutes		gwwtesting_db-worker_1
cbdae3b6d462	gameswithwords/api:latest	"bash start.debug.sh"	5 minutes ago	Up 5 minutes	3000/tcp	gwwtesting_api_1
2df725924d0a	gameswithwords/listener-quiz:latest	"bash start.debug.sh"	5 minutes ago	Up 5 minutes		gwwtesting_listener-quiz_1
411db4abcb82	gameswithwords/transactiondb:latest	"docker-entrypoint..."	5 minutes ago	Up 5 minutes	0.0.0.0:5433->5432/tcp	gwwtesting_transactiondb_1
274b6d3135eb	rabbitmq:management	"docker-entrypoint..."	5 minutes ago	Up 5 minutes	4369/tcp, 5671-5672/tcp, 15671/tcp, 25672/tcp, 0.0.0.0:8080->15672/tcp	gwwtesting_message-queue_1
33eca4069fdc	postgres:latest	"docker-entrypoint..."	5 minutes ago	Up 5 minutes	0.0.0.0:5432->5432/tcp	gwwtesting_db_1

and run the following command:

```
$ docker exec -it CONTAINER_ID bash
```

Followed by:

```
$ npm run migrations
```

At this point, you have created your database tables. You can check your Postgres manager to make sure your migrations were successful.

The stimuli for the quiz have been formatted and saved as a .csv file in **db-worker** -> **seeds** -> **short-quiz**. To seed the stimuli table in the database, you can run the following command in the db-worker container bash shell:

`$ node seeder.js NAME_OF_QUIZ` Insert the name of the quiz!

A series of questions will appear; the answers to all of them should be **Yes**.

```
root@01e49dd1b200:/app# node seeder.js listener-quiz
? Are you in pushkin-db docker? Yes
? Have you ran your migrations? Yes
? Have you properly filled out your seed CSV files? Yes
```

Hint: Every time you make changes to files you will need to rebuild the Docker containers, since the changes do not get updated automatically. To do this, run the following command: Every time you make changes to files you will need to rebuild the Docker containers, since the changes do not get updated automatically. To do this, run the following command:

```
$ docker-compose -f docker-compose.debug.yml up --build
```

Testing The Front End

Your are now ready to test your experiment locally. Once again, you need to install the dependencies in **front-end**. Here is how to do it:

- Change the working directory to **front-end** (`cd front-end`) and run the following commands:

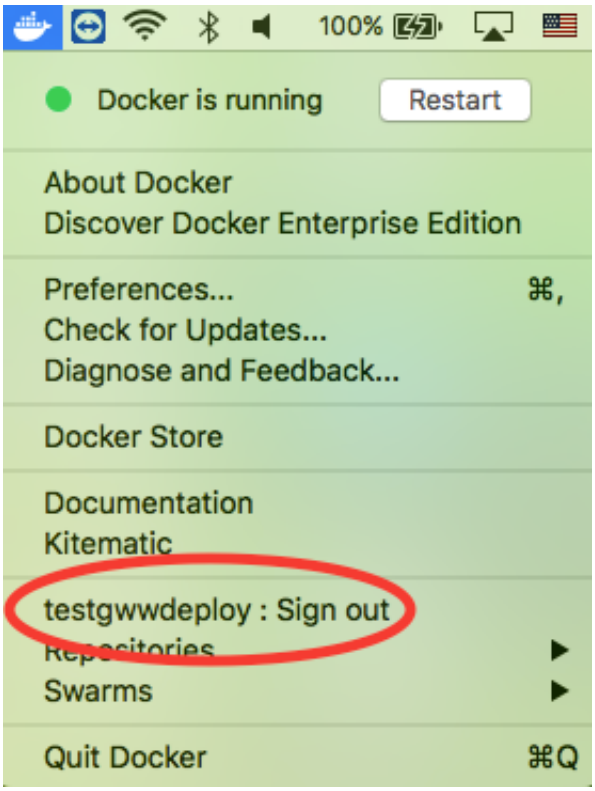
```
$ npm install

$ npm start
```

At this point you should be able to see the front end of the website in your browser.

Live Deployment

Once you have confirmed that your website works locally, you can begin live deployment. At this point, you should have a working Dockerhub ID. Ensure that you are signed in to Dockerhub on your computer before proceeding.



Amazon Web Services (AWS)

Open your AWS account, and go to **IAM** -> **Users**. You need to add a new user that will later be used by Rancher. It should have Programmatic Access and the following permissions:

AmazonEC2FullAccess
AmazonRoute53FullAccess

Once the user is created, you need to download the csv file with the security credentials which will be used later.

You will also need an S3 bucket to store your website's static content. Go to **Storage** -> **S3**. Create a bucket with the following configuration:

Bucket name: YOUR_BUCKET_NAME Insert the name of the bucket!
Region:US East (N. Virginia)
Manage public permissions: Grant public read access to this bucket

This will allow for your website's content to be accessible. Once the bucket is created, click on it and select **Permissions** -> **Bucket Policy**. Use the following policy:

Important: Replace BUCKET_NAME with your bucket's name in the configuration below!

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadForGetBucketObjects",
            "Effect": "Allow",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::BUCKET_NAME/*"
        }
    ]
}
```

You will also need to change the CORS configuration to:

```
<?xml version="1.0" encoding="UTF-8"?>
<CORSConfiguration xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
<CORSRule>
    <AllowedOrigin>*</AllowedOrigin>
    <AllowedMethod>GET</AllowedMethod>
    <MaxAgeSeconds>3000</MaxAgeSeconds>
    <AllowedHeader>Authorization</AllowedHeader>
</CORSRule>
</CORSConfiguration>
```

After saving the S3 bucket settings, open the file **run.js** located in the **front-end** folder. On line 112, change the bucket name to the name of the bucket you just created:

```
 s3Params: { Bucket: 'YOUR_BUCKET_NAME' },
```

Amazon CloudFront is a service that speeds up distribution of your static and dynamic web content, such as .html, .css, .js, and image files, to your users. You will create a CloudFront distribution for your website.

Go back to **AWS** -> **Cloudfront** -> **Create Distribution** -> **Web**. Use the following settings:

- Origin Domain Name: YOUR_BUCKET.s3.amazonaws.com Insert the name of your bucket here!
- Origin ID: S3-YOUR_BUCKET Insert the name of your bucket here!
- Viewer Protocol Policy: HTTP and HTTPS and **Create Distribution**.

Click on the newly created distribution and go to **Behaviors** -> **Edit** -> **Cache Based on Selected Request Headers** -> **Whitelist** From the Filter Headers options select **Origin** -> **Add** and then select **Yes, Edit** on the bottom of the settings page to finish.

From the **Cloudfront Distributions** page take the domain name of your distribution and insert it on line 22 of **run.js**:

Insert the name of your distribution below!

```
url: 'https://YOUR_DISTRIBUTION.cloudfront.net/',
```

Next, open **/front-end/core/baseUrl.js** and replace the CloudFront URL with the new one:

Insert the name of your distribution below!

```
const baseUrl = process.env.NODE_ENV === 'production' ? 'https://YOUR_CLOUDFRONT.net'
 : '';
```

Next, open front-end/webpack.config.js and edit the cloudfront distribution on line 45:

Insert the name of your distribution below!

```
publicPath: isDebug ? '/dist/' : "//YOUR_DISTRIBUTION.cloudfront.net/dist/",
```

Building and tagging Docker images

You will need to build, tag and push the Docker images for your live deployment to Dockerhub, where they will exist as Docker repositories that can be used to recreate your application on multiple hosts.

In a new Terminal window, `cd` into `front-end` and run the following script (editing the relevant parts first):

Insert your Dockerhub ID!

```
env NODE_ENV=production npm run publish &&
cd .. &&
cp -rf ./front-end/public/**.ico ./server/html &&
cp -rf ./front-end/public/**.txt ./server/html &&
cp -rf ./front-end/public/**.html ./server/html &&
cp -rf ./front-end/public/**.xml ./server/html &&
docker build -t DOCKERHUB_ID/api:latest api &&
docker build -t DOCKERHUB_ID/cron:latest cron &&
docker build -t DOCKERHUB_ID/db-worker:latest db-worker &&
docker build -t DOCKERHUB_ID/server:latest server &&
docker build -t DOCKERHUB_ID/short-quiz:latest workers/short-quiz &&
docker push DOCKERHUB_ID/api &&
docker push DOCKERHUB_ID/cron &&
docker push DOCKERHUB_ID/db-worker &&
docker push DOCKERHUB_ID/server &&
docker push DOCKERHUB_ID/short-quiz
```

The purpose of the script is to copy relevant files to the server folder of your project, build Docker images and tag them. The tags can be used in case you want to have several versions of an image uploaded to Dockerhub. You can specify which image to use in production by specifying the image's tag.

Security Groups

Your databases and Rancher instance will need special security groups with the right permissions. You can create them in your **AWS** account. Go to **EC2** -> **Security Groups** -> **Create Security Group**

1. Database security group (DATABASE_SECURITY_GROUP)
   Rules:
   MySQL/Aurora - TCP - 3306 - Anywhere
   PostsgreSQL - TCP - 5432 - Anywhere

2. Rancher security group (RANCHER_SECURITY_GROUP)

Rules:
HTTP - TCP - 80 - Anywhere
HTTPS - TCP - 443 - Anywhere
Custom TCP - TCP - 8080 - Anywhere
SSH - TCP - 22 - Anywhere

Creating Databases

You will need to create the databases for your project through AWS.

Write all the credentials down! You will need most of them later!
Go back to **AWS** -> **Services** -> **RDS** -> **Launch DB Instance**

1) **Amazon Aurora MySQL** (for Rancher)

- Instance specifications:

  DB instance class: db.t2.small
  Multi-AZ deployment: No
  DB instance identifier: YOUR_IDENTIFIER Insert your identifier!
  Master Username: YOUR_USERNAME (will be used to log in to the database) Insert your username!
  Master Password: YOUR_PASSWORD Insert your password!
  VPC Security groups: Select existing VPC security groups: DATABASE_SECURITY_GROUP
  DB Cluster Identifier: YOUR_IDENTIFIER Insert your identifier!
  Database name: YOUR_DATABASE_NAME Insert your database name!
  Database port: 3306
  Backup retention period: 7 days
  Maintenance window: Select window: SELECT_A_WINDOW

  At this point you are ready to click on **Launch Instance** and move on to creating the other databases.

2) **PostgreSQL** (main database)

- Instance specifications:

  DB instance class: db.t2.medium
  Multi-AZ deployment: No (for the workshop) Yes (if you are doing this to deploy a real website)
  Allocated storage: 100GB
  DB instance identifier: YOUR_IDENTIFIER Insert your identifier!
  Master Username: YOUR_USERNAME (will be used to log in to the database) Insert your username!
  Master Password: YOUR_PASSWORD Insert your password!
  VPC Security groups: Select existing VPC security groups: DATABASE_SECURITY_GROUP
  Database name: YOUR_DATABASE_NAME Insert your database name!
  Database port: 5432
  Backup retention period: 7 days
  Maintenance window: Select window: SELECT_A_WINDOW

3) **PostgreSQL** (transactions database)

- Instance specifications:

  DB instance class: db.t2.small

Multi-AZ deployment:No (for the workshop) Yes (if you are doing this to deploy a real website)
Allocated storage: 100GB
DB instance identifier: YOUR_IDENTIFIER Insert your identifier!
Master Username: YOUR_USERNAME (will be used to log in to the database) Insert your username!
Master Password: YOUR_PASSWORD Insert your password!
VPC Security groups: Select existing VPC security groups: DATABASE_SECURITY_GROUP
Database name: YOUR_DATABASE_NAME Insert your database name!
Database port: 5432
Backup retention period: 7 days
Maintenance window: Select window: SELECT_A_WINDOW

When you are done creating the databases, go back to the list of **Instances** and get the endpoints for the databases you created by clicking on them and scrolling down to **Connect** -> **Endpoint**.

Configuring *docker-compose.production.yml*

You need a Docker file (***docker-compose.production.yml***) that describes your production environment similar to the YAML file used to describe the development environment. A YAML file containing the necessary Pushkin elements already exists, and all you need to do is add your new experiment in the following format:

Insert your Dockerhub ID, database URL and transactions database URL!

```
short-quiz:
    image: 'DOCKERHUB_ID/short-quiz:latest'
    command: bash start.sh
    depends_on:
      - message-queue
    environment:
      - 'AMPQ_ADDRESS=amqp://message-queue:5672'
      - QUEUE=short-quiz
    links:
      - message-queue
```

and change the following:

- For every image, change the DOCKERHUB_ID to your Dockerhub ID;
- Insert the DATABASE_URL and TRANSACTION_DATABASE_URL (the format for those is: **postgres://user_name:password@host:port/database_name**), they appear twice - under environment for **cron** and for **db-worker**).

Hint: Host refers to the endpoint for the database in question.

Creating a Rancher Instance on AWS

Rancher is a container manager that gives you the tools to easily manager your Docker containers in production. AWS offers the option to create a Rancher EC2 instance. We will use an instance like that to deploy Pushkin.

First, you will need to create the instance. Go back to your **AWS account** ->**EC2**-> **Launch Instance** -> **AWS Marketplace** -> **Search** -> **Rancher** and select the first option.

- Instance configuration:
  Type: t2.medium
  Enable termination protection: Yes (Protect against accidental termination)

  Size: 16GiB

  Add Tag: key=Name, value=NAME_OF_RANCHER_INSTANCE Insert a name for your instance!
  Security Group: Select Existing: RANCHER_SECURITY_GROUP
  Key Pair: ec2_workshop (for this workshop, otherwise, your choice, but make sure you have access to the private key for the pair you choose!)

  Launch the instance!

Elastic IP

This step is optional, but it is helpful to assign an Elastic IP to your instance so that the IP remains constant.

In your AWS account, go to **EC2** -> **Elastic IP** -> **Allocate new address** -> **Allocate**

Once it has been created, select it, and from the **Actions** menu select **Associate address**. Select your Rancher instance and then **Associate**

Find Route 53 in your AWS account. Your domain name should appear under **Hosted Zones**. Click on it and then on **Create Record Set**:

Name: YOUR_RANCHER_INSTANCE_NAME.DOMAIN_NAME (ex. workshop-rancher.jenningsdeveloperworkshop.com)
Value: YOUR_RANCHER_INSTANCE_IP Insert your Rancher instance's IP!

Associating your Rancher IP with your domain name will allow you to easily access the GUI for Rancher later on without having to retrieve the IP every time.

Setting Up the Rancher Instance

You will be using the Nginx web server. For the purposes of the workshop, we will not use a real SSL certificate. Instead, we will use expired SSL certificates previously used for gameswithwords.org to practice how to configure SSL. **nginx.conf** is located in the **short-quiz** repository you cloned earlier.

You will need to open **nginx.conf** in a text editor and edit the following:

line 12: server-name YOUR_RANCHER_INSTANCE_NAME.domain_name
(ex. workshop-rancher.jenningsdeveloperworkshop.com)

line 35: server-name YOUR_RANCHER_INSTANCE_NAME.domain_name
(ex. workshop-rancher.jenningsdeveloperworkshop.com)

However, you might want to use active SSL certificates if you decide to deploy a real Pushkin project. Here is how to change your SSL certificate configuration in that case:

nginx.conf is located in the **pushkin_workshop** folder. First, you will need to open **nginx.conf** in a text editor and edit the following:

line 12: server-name YOUR_RANCHER_INSTANCE_NAME.DOMAIN_NAME
line 13: ssl_certificate /etc/nginx/ssl/NAME_OF_YOUR_CERTIFICATE.crt
line 14 ssl_certificate_key /etc/nginx/ssl/NAME_OF_YOUR_KEY.key **line 35:** server-name YOUR_RANCHER_INSTANCE_NAME.DOMAIN_NAME

NOTE: If you are deploying a real Pushkin project, you will need to generate the SSL certificates before this step, and you can use a provider of your choice (we use namecheap.com). For the purposes of the workshop, you will use SSL certificates we have previously generated for gameswithwords.org.

After adjusting **nginx.conf** you will need to complete the following steps:

1. Connect to the Rancher instance

I recommend using an SSH client such as Termius to connect to the instance, but you can also SSH to your instance from Terminal on your Mac (`ssh –i path_to_key rancher@IP_OF_INSTANCE`). Make sure you have the key from the key pair you specified when creating the Rancher instance.

NOTE: The IP of the Rancher instance will be the Elastic IP you just associated with it.

Create a directory named rancher (`mkdir rancher`) and `cd` into it.

You will need to transfer the **nginx.conf** file, the **.crt** file, and the **.key** file to the Rancher instance and place them in the rancher directory. You can use an FTP client such as FileZilla or use scp.

You will need to execute the following commands:

```
sudo docker run -d --restart=unless-stopped --name=rancher-server -p 8080:8080 ranche
r/server --db-host DB_URL --db-port 3306 --db-user DB_USER --db-pass DB_PASS --db-nam
e DB_NAME
```

NOTE: Do not forget to change the DB_URL, DB_USER, DB_PASS and DB_NAME to the values for your **Amazon Aurora** database created earlier. Followed by:

```
sudo docker run -d --restart=unless-stopped --name=nginx -v ~/rancher/nginx.conf:/etc
/nginx/conf.d/default.conf -v ~/rancher/gameswithwords.crt:/etc/nginx/ssl/gameswithwo
rds.crt -v ~/rancher/gameswithwords.key:/etc/nginx/ssl/gameswithwords.key -p 80:80 -p
 443:443 --link=rancher-server nginx
```

You should have a running Nginx container at this point. You can use `docker ps` to check and make sure you have a running container.

Important: If you encounter any error messages, use the following command:

```
docker stop $(docker ps -a -q) && docker rm $(docker ps -a -q) && docker rmi $(docker
 images -q)
```

to stop all containers, remove all containers and remove all images associated with them. When all containers have been removed, repeat:

```
sudo docker run -d --restart=unless-stopped --name=nginx -v ~/rancher/nginx.conf:/etc
/nginx/conf.d/default.conf -v ~/rancher/YOUR_CERTIFICATE_NAME.crt:/etc/nginx/ssl/YOUR
_CERTIFICATE_NAME.crt -v ~/rancher/YOUR_KEY_NAME.key:/etc/nginx/ssl/YOUR_KEY_NAME.key
 -p 80:80 -p 443:443 --link=rancher-server nginx
```

You will also need to run:

```
docker exec –it nginx bash
service nginx restart
```

Restarting Nginx ensures that it's functioning properly before exiting the container.

NOTE: You can use CTRL+D to get out of the instance after executing the last command.

Rancher

At this point, if everything worked well, you can open a browser of your choice and point it to the IP of your Rancher instance (The Elastic IP you gave it earlier should be displayed in your AWS account under EC2 next

to the name of the instance) or the alias (YOUR_RANCHER_INSTANCE_NAME.DOMAIN_NAME). You should not get any error messages and the Rancher GUI should be available to you.

By default, Access Control in Rancher is not configured. Anyone who has the IP of your Rancher instance can access the Rancher GUI. It is highly recommended that you set up access control. This is how you do it:

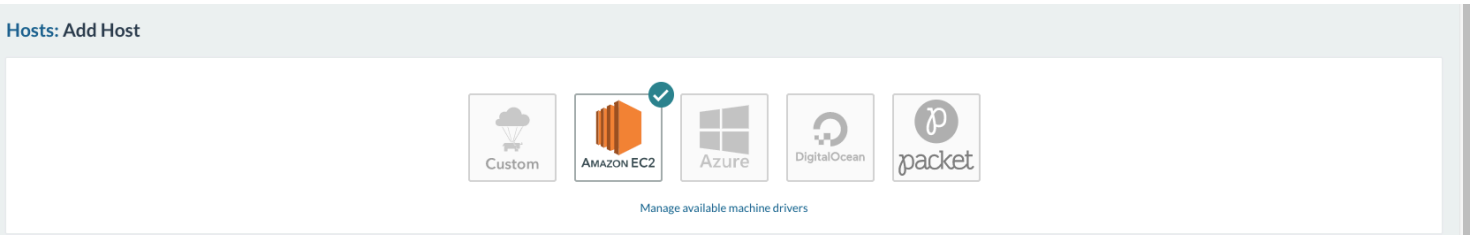
Admin -> **Access Control** -> **Local**

Enter the credentials you would like to use and click on **Enable Local Auth**

Next, you will need to create your first host to get your web application up and running. Go to **Infrastructure** -> **Hosts** -> **Add a Host**

You will need to select a **Host Registration URL**; choose **Something else**, copy the address from "This site's address" above, edit it from https to **http** and add **:8080** to the end of it.

On the next page, choose **Amazon EC2:**



and fill out the remaining settings as follows (use the access key and secret key from the AWS Rancher user credentials you downloaded earlier):

Region: us-east-1
Access key: Your AWS access key
Secret key: Your AWS secret key
VPC/Subnet: vpc
Security group: Custom: Choose an existing group: rancher-machine
Name: YOUR_HOST_NAME Choose a name for your host!
Instance type:t2.medium
SSH User: rancher.
AMI: Rancher OS AMI List: copy the AMI for us-east-1

To check if the host is being created after you click on **Create**, go to **Infrastructure** -> **Hosts** and you should be able to see your new host.

The Transactions Database

You will need to create a table in the transactions database. Open your Postgres manager. This is done separately only once during initial deployment. Enter the following:

Server host: YOUR_DATABASE_ENDPOINT Change to actual database endpoint!
Login: DATABASE_USERNAME Change to actual database username!
Password: YOUR_DATABASE_PASSWORD Change to actual database password!

Once you are connected to the transactions database, you will need to go back to the **pushkin_workshop** folder and find a script called **CreateTransactionDB.sql**. Copy the contents of this script and run them in the Postgres manager. This step will create the table for your transactions.

More on Rancher

In the **pushkin_workshop** folder, find **rancher-compose.yml**. Rancher Compose is a multi-host version of Docker Compose. Containers started by Rancher Compose will be deployed on all the hosts you create. If you have added new quizzes, you will need to add them to the end of the file following this format:

```
NAME_OF_QUIZ:
    scale: 1
    start_on_create: true
```

If you need to add an SSL certificate:

Go back to the Rancher GUI in your browser and go to: **Infrastructure** -> **Certificates** -> **Add Certificate**

You will need to provide the following (these are the same as the ones you used when setting up the Rancher instance):

Name: YOUR_CERTIFICATE_NAME Choose a name!
Private key: YOUR_PRIVATE_KEY Use: gameswithwords.key
Certificate: YOUR_CERTIFICATE Use: gameswithwords.crt
Chain Cert: your .ca-bundle file Use: gameswithwords.ca-bundle

You will also need to add a new stack in Rancher. A stack is a group of services. We will use a stack to group together all the services needed for a Pushkin project. Go to **Stacks** -> **User** -> **Add Stack**

Enter the following:

Name: YOUR_STACK_NAME Choose a stack name!
Optional: docker-compose.yml: use the contents of docker-compose.production.yml (located in the **pushkin_workshop** folder)
Optional: rancher-compose.yml: use the contents of rancher-compose.yml (located in the

pushkin_workshop folder)

You will need to upgrade the load balancer as soon as it is created.

Once all the services become active, you will need to run your migrations and seed your database tables in live mode. To do that, click on **db-worker** and from the options select **Execute Shell**. Run the following commands in the shell:

```
npm run migrations
node seeder.js short-quiz
```

You should repeat the second command for as many quizzes as you have. If you see a list of your stimuli it means that you were successful.

Setting up DNS in Rancher

You want to use the load balancer to distribute traffic to your application to the different hosts that are currently active on Rancher. To do that, you need to use your domain name as an alias for the load balancer. Once a user types in your domain name, they will be handled by the load balancer and sent to the host that currently has the lowest level of activity.

Under **Catalog** find **Route 53** and click on **View Details**. Enter the AWS Secret Access Key and AWS Access Key ID for the Rancher user created earlier. Enter the following:

Hosted Zone Name: YOUR_DOMAIN_NAME Use your domain name!
Hosted Zone ID: YOUR_HOSTED_ZONE_ID (found on **AWS** under **Route 53**) Use your hosted zone ID

You will also need to create a new Record Set on Route 53 under your domain name. The **Name** should be your domain, and the **Value** should be the URL of the load balancer.

Setting up DNS in AWS

In your AWS account, go to **Route 53** -> **Hosted Zones**

Choose your domain and **Create New Record Set**

In the Name field add **load-balancer** in front of your domain (**load-balancer.jenningsdeveloperworkshop.com**), and select **No** for Alias.

Create a second record set without adding anything before your domain name, choose **Yes** for Alias, and use **load-balancer.YOUR_DOMAIN_NAME** in the Value field.

Autoscaling - Optional and not needed for the workshop!

You might want to set up autoscaling if you would like to have hosts created when your web application gets more traffic, and have those hosts deactivated once they are not needed anymore. One way to set up autoscaling is to use Datadog to work along with Rancher.

In Rancher, go to **API** -> **Webhooks**

You can create webhooks to scale up and down. The type for those should be the name of your Rancher host.

Next, from **Catalog** find Datadog and launch.

You will also need to create a Datadog account and set up autoscaling rules using it.