Quiz App / DevOps (V3)

Activity 1: Fork + Clone from GitHub

1. Make a new directory on your PC for the source code

   ```
   mkdir c:\source
   ```

2. Sign into **github.com** using your GitHub account
3. Search for the repository **qa-apprenticeships/quiz-api**
4. In the top-right of the screen, click **Fork**
5. This should make a repository called **quiz-api** in your own GitHub account (for example github.com/fredsmith1970/quiz-api)
6. Repeat the process, to make your own fork of the repository **qa-apprenticeships/quiz-web**
7. cd into your source directory

   ```
   cd c:\source
   ```

8. Clone both the **quiz-api** and **quiz-web** repositories to disk

   ```
   git clone https://github.com/<yourname>/quiz-api
   git clone https://github.com/<yourname>/quiz-web
   ```

9. Launch a copy of Visual Studio Code and inside it open the **quiz-api** folder
10. Launch a second copy of Visual Studio Code and inside it, open the **quiz-web** folder

Activity 2: Get app running locally

1. Switch to instance of Visual Studio Code containing **quiz-api**
2. Open a new terminal (Terminal > New Terminal)
3. Restore the dependencies of the API application

   ```
   npm install
   ```

4. Start the API server running, listening for incoming HTTP REST API calls - by default, the API server will be listening on http://localhost:8001/

   ```
   npm start
   ```

5. Switch to the instance of Visual Studio Code containing **quiz-web**
6. Open a new terminal (**Terminal** > **New Terminal**)
7. Restore the dependencies of the web application

   ```
   npm install
   ```

8. Start the web server running, listening for requests for HTML pages and related assets - by default, the web server will be listening on http://localhost:8000/

   ```
   npm start
   ```

9. Open a browse tab, and navigate to **http://localhost:8000/** - you should see a page inviting you to join a quiz
10. Open another browser tab, and navigate to **http://localhost:8000/** - you should see a page inviting you to join a quiz
11. Open a third browser tab, and navigate to **http://localhost:8000/host** - you should see a page inviting you to pick a quiz to host
12. Try out the quiz, using the three browser tabs - the first two tabs represent two players, and the third tab represents the host running the quiz
13. Make a change to the code (such as the background colour of the website defined by **app.css**) and observe the difference - this shows we can try out our changes out locally first

Activity 3: Add restriction for 2 players minimum

Currently, the quiz can be started as soon as at least one player has joined. We'd like to change this so that you must have at least two players before the quiz can start.

1. Switch to the **quiz-api** instance of Visual Studio Code
2. Locate the lines of code within **src/services/quiz.js** that prevent the quiz from starting if there are no players yet
3. Change this code so the quiz will be prevented from starting if there are **less than two** players
4. Re-start the API service, and manually check that you can't start a quiz until you have at least two players

Activity 4: Unit testing, including new min 2 players rule

1. Run the existing API unit tests, in the terminal - they should all pass

   ```
   npm run test
   ```

2. Find the unit test within tests/quiz.nextStageAsync.js that tests "it should reject request if no players yet"
3. Copy and paste this test (the "it" block), then amend the copy to test that "it should reject request if only one player", and change the arrange part of the test to set up one player:

   ```
   it('should reject request if only one player', async function () {
       instanceBefore.players = [ {name: "Player 1"} ]; // One players
       await expect(quiz.nextStageAsync('1234')).to.eventually.be.rejectedWith('Not enough players yet'
   );
       expect(saveInstance.notCalled).to.be.true;
   });
   ```

4. Re-run all the tests - they should all pass

   ```
   npm run test
   ```

5. Prove the test would detect if the feature broke (regression testing), by changing the code in **src/services/quiz.js** back to its original version. Re-run the tests - you should see a failure

   ```
   npm run test
   ```

6. Put the code in **src/services/quiz.js** back to how it should be. Re-run the tests - should all pass

   ```
   npm run test
   ```

Activity 5: Deploy API (manually) to AWS Elastic Beanstalk

1. To prepare the upload file, you need to zip up the **src** folder and the **package.json** file from you API root folder, into a new zip file
2. Sign into AWS Console using your AWS credentials

   ```
   https://aws.amazon.com/console/
   ```

3. Start a new Elastic Beanstalk environment (EBS - environments - new environment)
   - Environment tier: **web server**
   - Application name: **quiz-api-XYZ** (your initials)
   - Domain: **quiz-api-XYZ** (your initials)
   - Platform: **managed platform > node.js**
   - Platform branch: (leave as default)
   - Platform version: (leave as default)
   - Application code: **upload your code**
   - Version label: **quiz-api-XYZ-source**
   - Upload the zip file as the application code
   - Tell AWS to go ahead and create the environment
4. Or - if no option to create a new environment, only the option to create a new application, then...
   - Start a new application

- Application name: **quiz-api-XYZ** (your initials)
- Platform: **Node.js**
- Platform branch: (leave as default)
- Platform version: (leave as default)
- Application code: **upload your code**
- Version label: **quiz-api-XYZ-source**
- Upload the zip file as the application code
- Click **configure more options** (really important!)
- Click **custom configuration**
- Tell AWS to go ahead and create the environment

5. Or - create an app with default environment first, then create the real environment (and terminate the old one)
   - Pick the application
   - Click Create environment
   - Environment name: **Quiz API - Production**
   - Domain name: **quiz-api-XYZ**
   - Platform: **Node.js**
   - Platform branch: (leave as default)
   - Platform version: (leave as default)
   - Application code: (pick existing code for application)
   - Tell AWS to create the environment
   - After the new environment is created
6. Wait while everything is created and the API is deployed
7. Check the deployment, using the URL below - should get back some JSON

   `https://quiz-api-XYZ.eu-west-2.elasticbeanstalk.com/quiz`
8. Explore the information about the environment, such as the auto-scaling options set by default

Activity 6: Configure health monitoring of API

1. Observe that querying **https://quiz-api-XYZ.eu-west-2.elasticbeanstalk.com/** returns a 404 error, because it is not a valid API endpoint as defined by our API code
2. Observe the health monitoring status of **Severe**, because this is the URL being polled to check the service
3. Navigate to **Configuration** > **Load Balancer** > **Edit**
4. Scroll to **Processes**
5. Tick then edit the environment process
6. Change the **Path** from **/** to **/quiz**
7. Click **Save**
8. Click **Apply** at the bottom (easy to forget!)
9. Shortly afterwards, observe the health status revert to **OK**
10. Try requesting different API URLs, and observe the change in the Monitoring statistics (can be a slight lag)

Activity 7: Automate the build + deployment of the API

We're going to use an AWS CodePipeline to control the build and deployment process, so we can go from a commit to source control to a live deployment totally automatically.

First, we'll need a way to tell AWS CodePipeline what steps are involved, and what files need to be deployed. We do that with a "YAML" file.

1. Add a new file to root of the API project, called **buildspec.yml** - paste the following content into it:

```
version: 0.2
phases:
  install:
    runtime-versions:
      nodejs: 12
  pre_build:
    commands:
      - npm install
  build:
    commands:
      - npm build
      - npm test
artifacts:
  files:
    - src/**/*
    - package.json
```

2. Save the file, and commit your changes back into source control

```
git add .
git commit -m "intial yaml configuration"
git push
```

3. Navigate to AWS CodePipelines (from Services link)
4. Click **Create Pipeline**, and fill out the details below:
    - Name: **quiz-api-XYZ** (your initialis)
    - Accept all other defaults, and click **Next**
    - Source provider: **GitHub (Version 2)**
    - Connection: click **Connect to GitHub**
    - GitHub App connection name: **quiz-api-XYZ**
    - Click **Authorize AWS Connector for GitHub**
    - Click **Install a new app**
    - Repositories: **only select repositories**
    - Pick the **quiz-api** repository
    - Click **install**
    - Back on the *Connect to GitHub* page, click **Connect**
    - Back on the *Add source stage* page, continue filling in the form...
    - Repository name: **<your-github-name>/quiz-api**
    - Branch: **main**
    - Output artifact format: **CodePipeline default**
    - Click **Next**
    - In the *Add build stage* page, complete the following details...
    - Build provider: **AWS CodeBuild**
    - Project name: click **Create Project**
    - Project name: **quiz-api-XYZ**
    - Environment image: **Managed image**
    - Operating system: **Ubuntu**
    - Runtime: **standard**
    - Image: **aws/codebuild/standard:4.0**

- Image version: **Always use the latest image for this runtime version**
- Environment type: **Linux**
- Privileged: **No**
- Service role: **New service role**
- Build specifications: **Use a buildspec file**
- Buildspec name: **(leave blank)**
- Click **Continue to CodePipeline**
- Back on the *Add build stage* page, continue filling the form...
- Build type: **single build**
- Click **Next**
- In the *Add deploy stage* page, complete the form...
- Deploy provider: AWS Elastic Beanstalk
- Application name: **quiz-api-XYZ**
- Environment name: **quiz-api-XYZ**
- Click **Next**
- Click **Create Pipeline**

5. The pipeline you created should now start running, and go through all the stages from getting your source code, to building the deployment bundle, to deploying it into your Elastic Beanstalk environment
6. To test this really works, make a change to the API source code, such as on line 35-40 of **src/services/quiz.js** making the quiz names names get returned as upper case

```
const list = quizzes.map(quiz => {
    return {
        id: quiz.id,
        name: quiz.name.toUpperCase()
    };
});
```

7. Save the file, and commit to source control - and watch the updated API application get deployed

```
git add .
git commit -m "players names uppercase"
git push
```

8. Check the change has worked, by navigating to your API hosted in AWS, and using the **/quiz** URL - the quiz names should now be upper case
9. Finally, check that if the tests are run as part of the deployment, and that if they fail, the deployment is stopped. Edit the source code to stop enforcing the "2 players minimum" rule (but keep the test there) - and push the change to source control. When you view the build progress, you should see the build & deploy fail.
10. (Fix the code and deploy it again, before you leave this step.)

Activity 8: Explore options around deployment strategies

1. Within AWS Elastic Beanstalk, click on **Configuration** > **Managed updates** > **Edit**

Activity 9: Set up an HTTPS/SSL certificate for the hosted API URL

This is really painful, but only because we're tyring to do everything without incurring charges. For a real company using real custom domains, this is much simpler.

1. Register with the **sslforfree.com** website (email and password)
2. Request an SSL certificate for your API domain
    - Domain name = domain URL of your API, e.g. quiz-api-ajb.eu-west-2.elasticbeanstalk.com
    - Validity = 90 days (free)

- Auto-generate CSR = yes
- Price = $0 (free option)

3. Complete the verification steps
   - Choose HTTP File Upload verification
   - Download the auth file
   - Copy the auth file into the **src/pki-validation** folder within the API project
   - Commit and push changes to git
   - This should republish the API on AWS
   - Test the validation URL in in the cloud **/.well-known/pki-validation/<filename.txt>**
   - Back in the **sslforfree** website, click "verify domain"
4. Install the certificate in AWS
   - Download the certificate zip file
   - Expand the zip file to get access to the individual files within
   - In AWS, go to **Certificate Manager**
   - Click **import a certificate**
   - For **certificate body**, copy and paste the text contents of the **certificate.crt** file
   - For **certificate private key**, copy and paste the text contents of the **private.key** file
   - For **certificate chain**, copy and paste the text contents of the **ca_bundle.crt** file
   - **Finish** importing the certificate
5. Use the certificate for HTTPS connections to the API application
   - Open the API environment in AWS
   - Go to **configuration / load balancer / edit**
   - Under **listeners**, add a new listener
     - Port = **443**
     - Protocol = **HTTPS**
     - SSL certificate = (the one you just installed)
   - At the bottom of the page, click **apply**
6. Verify that you can now access your API through an HTTPS URL, in addition to the original HTTP URL

Activity 10: Connect local front-end to cloud back-end

1. Edit the **src/services/config.service.js** file to change the **service.apiBase** variable to the URL of your API in AWS, e.g. **https://quiz-api-ajb.eu-west-2.elasticbeanstalk.com/**

   NB: Make sure you include a trailing slash on the end
2. Run the web application locally again, and check you can host a quiz

   ```
   npm start
   Browse to -- http://localhost:8000/host
   ```
3. When you're happy it works, commit your web app changes to git

   ```
   git add .
   git commit -m "changed back-end URL"
   git push
   ```

Activity 11: Deploy front-end website to AWS Amplify (manually)

1. Prepare a ZIP file of the assets to deploy, by zipping up the **contents** of the **src** folder into a zip file (i.e. so that the file **index.html** is in the root of the zip file, **not** within a folder called **src** within the root of the zip file)

2. Navigate to **Amplify** within AWS Console
3. Click Deploy / **Get Started**
    - Source: **deploy without git provider**
    - App name: **quiz-web-XYZ** (your initials)
    - Environment name: **prod** (production)
    - Method: **drag and drop**
    - Drag the zip file you created onto the upload area
4. Click **Save and deploy**
5. Configure Amplify app so any unmapped files resolve to index.html (without this step, Amplify won't work when you navigate anywhere other than the root of the web app) - we need to do this because the application is a single-page app (SPA) and everything happens through index.html
    - Go to **Rewrites and Redirects** section
    - Add a new rule
        - Source address: **/<*>**
        - Target address: **/index.html**
        - Type: **404 (rewrite)** [not redirect!]
    - Click **Save**
6. Browse to the web app URL, using the link in Amplify - should be able to use the application OK.
7. If communications with your server fail, check the JavaScript console for errors (F12) - do you see one saying HTTPS pages can't call HTTP APIs? If so, you'll need to switch your web app to connect to my (HTTPS) API URL
    - Edit your web code again to change the base URL of the API
    - Commit back into version control
    - Prepare a revised zip file and upload it
8. Try making a change to the app, such as the background colour of the pages (**app.css**) - commit to source control, but then rebuild the zip file and upload again

Activity 12: Automate the deployment of the web front-end to Amplify

First, we need to automate the preparation of a /**dist** folder for the files to be deployed - we need the tests to be deployed to the build server, but not to the actual Amplify CDN servers.

1. Add a build task in **package.json** to clear the /dist folder, then copy the contents of the **src** folder into the /**dist** folder

    ```
    "build": "rimraf \"dist\" && cd src && cpx \"**/*\" ../dist"
    ```

2. Save the file and test that the build works as expected, by running

    ```
    npm run build
    ```

    and making sure the contents of the **/dist** folder have been populated

Next, we need to prepare a YAML file to explain to Amplify how we want it to perform the build and deploy steps

3. Add a new file to the root of the web project, called **amplify.yml** - then paste in these contents:

    ```
    version: 1
    frontend:
      phases:
        preBuild:
          commands:
            - npm install
        build:
          commands:
            - npm run build
      artifacts:
        baseDirectory: /dist/
        files:
          - '**/*'
    ```

```
cache:
  paths:
    - node_modules/**/*
```

4. Save all the changes files, and commit your changes back into source control

```
git add .
git commit -m "prepared yaml file"
git push
```

Next, we set up AWS Amplify for automated deployment from Git changes

5. In Amplify, delete the current (manually-deployed) app - we need to recreate it if using a different source
6. Create new app with these settings...
   - Source: **GitHub**
   - [Go through process to authorise access to GitHub from AWS - once done it doesn't ask again?]
   - Repository: (your **quiz-web** repository)
   - Branch: **main**
   - App name: **quiz-web-XYZ** (your initials)
   - Under Build and test settings, the page should say **"We detected amplify.yml in your repository and will use it to deploy your app"**
   - Click **Next**
   - Click **Save and deploy**
   - Application should successfully go through the provision -> build -> deploy -> verify stage
   - Browse to the application's URL - should see application working OK
7. Change something about the app, for example the background colour via **app.css**, or some text on the main screen, the save your changes into source control - you should see the application build and deployment pipeline complete, and the new version of your app be deployed

Activity 13: Automating the testing of the front-end during build + deploy

1. Run the automated tests for the web front-end:

   ```
   npm run test
   ```

   Note how a web browser is launched (this is because the front-end JavaScript being tested has to run inside a browser engine).

   The tests also make use of "mocking" so they can test the website without it actually connecting to a real back-end API (which would be outside the control of the tests).

2. Set up a new NPM task in the **package.json** file suitable for when the build server runs the tests. This needs stop after running the tests, and also needs to use a "headless" version of Chrome

   ```
   "test-ci": "karma start karma.conf.js --single-run --browsers ChromeHeadlessNoSandbox"
   ```

3. Check that the tests still work using the headless browser:

   ```
   npm run test-ci
   ```

4. Add **npm run test-ci** as a command in the **amplify.yml** file, in the **postBuild** section (so the tests are run after the website has been built)

   ```
   postBuild:
     commands:
       - npm run test-ci
   ```

5. Finally, we need to ensure that Chrome (and its headless variant) are actually available on the build server - so add this to the **amplify.yaml** file:

   ```
   preBuild:
     commands:
       - npm install
       - wget --no-verbose https://dl.google.com/linux/direct/google-chrome-stable_current_x86_64.rpm
       - yum -y -q install google-chrome-stable_current_x86_64.rpm
   ```

6. Save all files and commit them back into source control, and you should see the automated build be triggered, and it run the web app tests as part of the deployment (stopping the deployment if the tests fail)

Activity 14: Deploy to different environments from different branches

1. Use git to create another branch from master: **dev**

```
git branch dev
git checkout dev
// Make changes, e.g. to app.css
git add .
git commit -m "message"
git push --set-upstream origin dev
```

2. In Amplify, pick the app then select "connect branch" and pick branch **(dev)**
3. Then apply and deploy - should now have a separate environment deployed for the dev branch, separate from the main (production) branch
4. To publish **dev** changes to **production**:

```
git checkout master
git merge dev
git add .
git commit -m "message"
git push
```

Or, use pull requests (from within GitHub)

Activity 15: Connect each environment's web app to the right API URL

1. Make a copy of the **config.service.js** file, calling it **config.service.js.template**
2. Replace the line that assigns the value to **service.apiBase** to use a placeholder instead, i.e.

```
service.apiBase = '${API_BASE}';
```

3. Install as a development dependency the NPM module **envsub**

```
npm install envsub --save-dev
```

4. Add an extra task to the **package.json** file, which can use the **envsub** program to regenerate the **config.service.js** file from the **config.service.js.template** file + current values of the **environment variables**

```
"substitute": "envsub ./dist/services/config.service.js.template ./dist/services/config.service.js"
```

5. Test this works by running in the terminal (make sure it's a **CMD** terminal)

```
npm run build
SET API_BASE=http://acme.com/
npm run substitute
```

Check the the file **dist/services/config.service.js** has been updated to use the new **acme** URL

6. Make the substitution task get run as part of the build task:

```
"build": "rimraf \"dist\" && cd src && cpx \"**/*\" ../dist && npm run substitute"
```

7. Save all files and commit them back into version control (make sure the **package.json** file is updated in all branches!)
8. In AWS Amplify Console, select the quiz app, then click on **Environment variables**, then click **Manage variables**
   - Add a variable called **API_BASE**, and give it the value **https://REPLACE/**
   - Add an override for the **main** branch, and give it the value matching the URL of the production API
   - Add another override for the **dev** branch, and give it a different URL
   - Trigger a build of the master branch, then check that the hard-coded API URL within the deployed website (in the master environment) is what you expect
   - Trigger a build of the dev branch, then check that the hard-coded API URL within the deployed website (in the dev environment) is what you expect