

How to Deploy the App

written by Elliot

PART.1 AWS

• 关于AWS账号

按照老师的教程我并没有注册到一个可用的正式的AWS账号，所以只有根据后半段教程创建了一个 **AWS Educate** 的账号，注册具体的过程可以参考PPT，需要注意的点就是申请邮箱要填学校邮箱，以及及时到邮箱里面点击验证邮件里面的链接。如果访问AWS比较慢的话建议使用交大VPN。

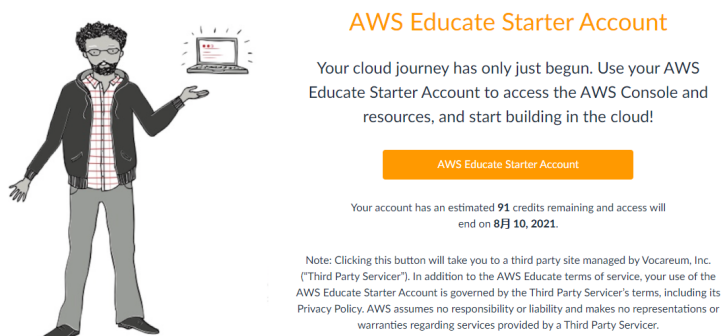
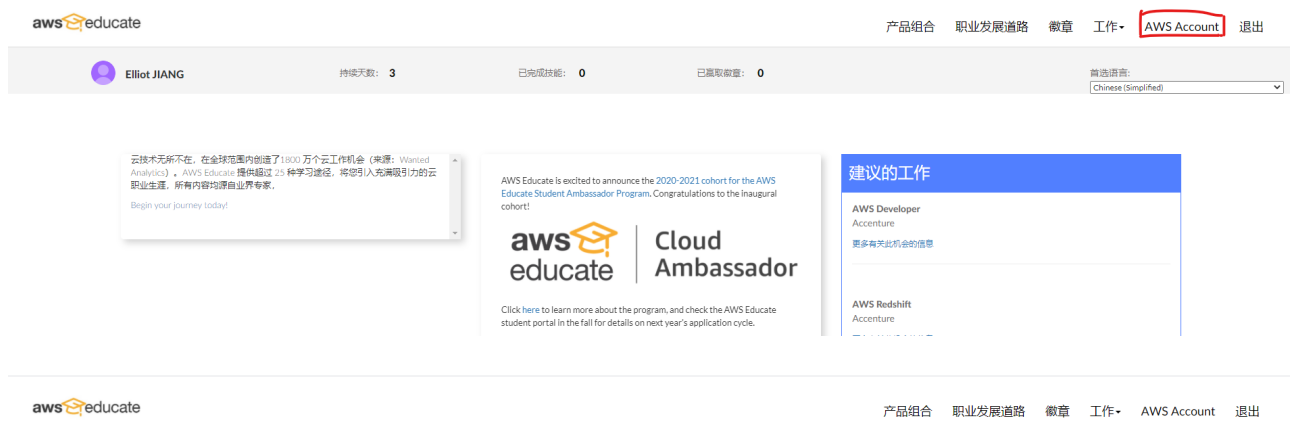
• 使用AWS Educate

AWS Educate的网址在这里 [AWS Educate](#)

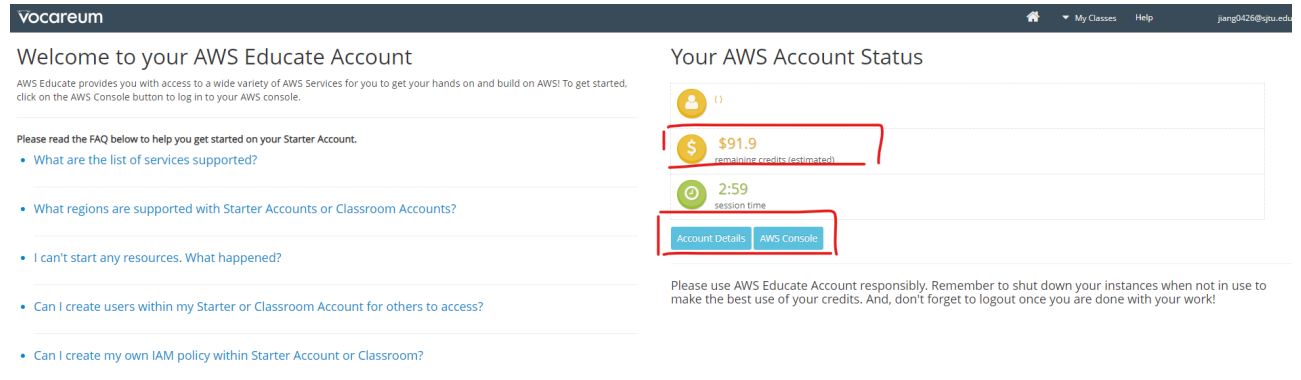
进去之后就可以看到注册和登录的链接，点击登录。



用注册并激活过的邮箱和密码登录，就会进入下面的页面，点击右上角的 **AWS Account** 接着点击中间的按钮



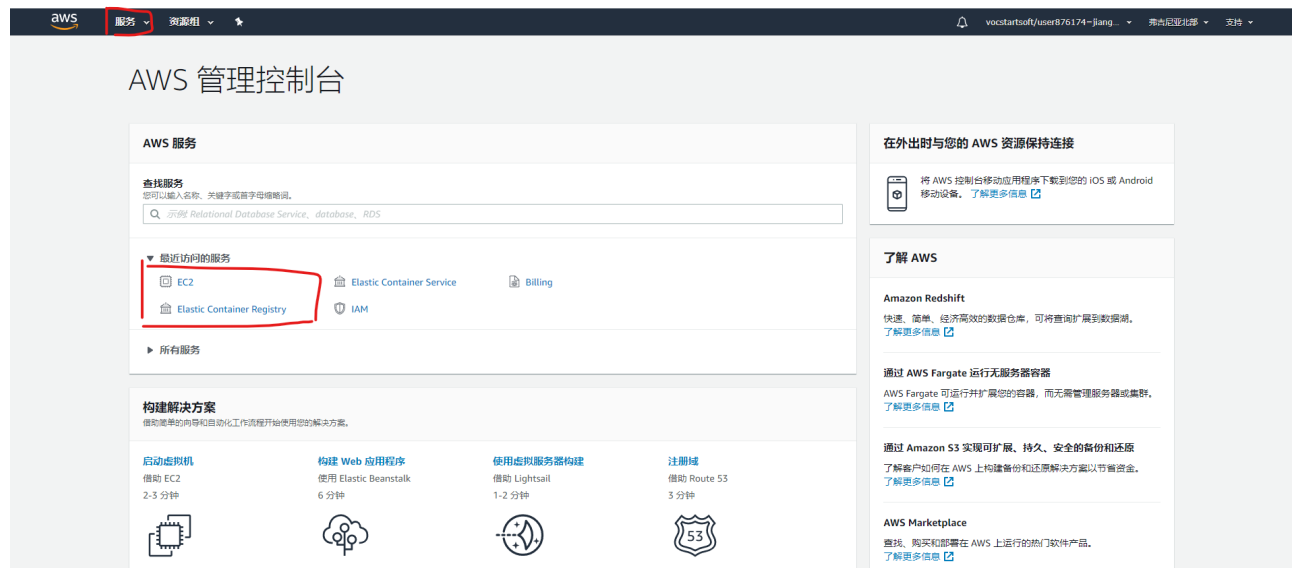
这个时候就会被送到第三方的平台（接下来称为 **Workbench**），通过这个平台可以查看当前账户剩余的费用，以及后面会用到的 **Account Detail**



Account Detail 可以查看AWS CLI的认证信息，**AWS Console** 可以转到AWS的控制台

- **AWS 控制台**

AWS控制台的界面如下：



目前我们的项目主要用到的服务是

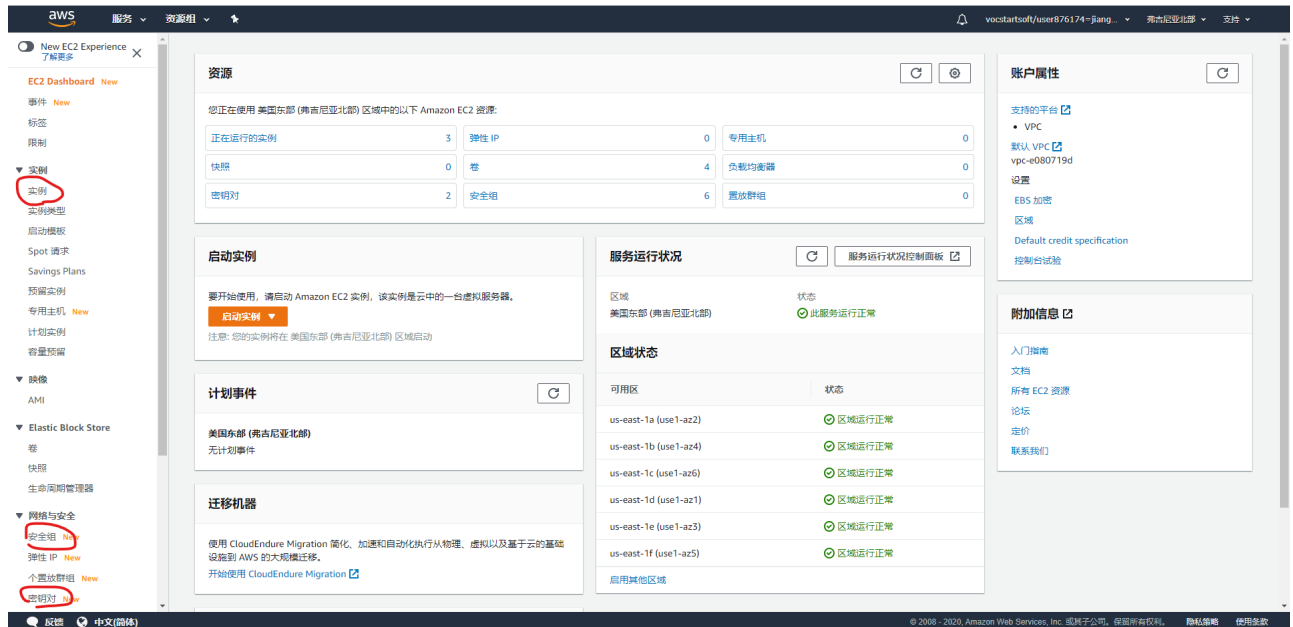
EC2 —— 搭建虚拟机来运行服务器和前端

Elastic Container Registry(ECR) —— 储存Docker镜像

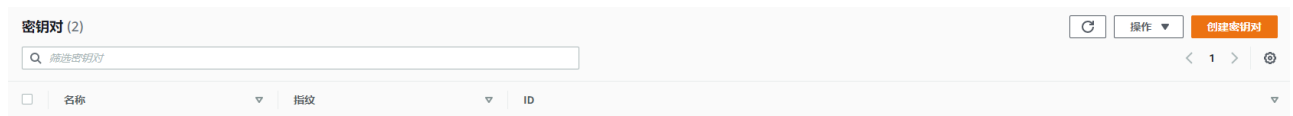
第一次可以在左上角的服务栏中找到

- **密钥对**

密钥对是用来通过本机连接到远程服务器的，公钥保存在云端（AWS），私钥保存在本机（PC），一直都会用到的。在 **EC2** 控制面板的侧边栏可以找到，点击 **EC2** 进入页面（第一次使用好像页面不一样，但是创建第一个实例时会让你选择密钥对，那个时候也可以新建密钥对）



点击新建密钥对



随便输一个名字，选择 **.pem格式**，点击新建，就会弹出一个下载保存的窗口，把密钥对的私钥保存在本地某个地方，一定要记好自己密钥对的保存位置和名字。

创建密钥对

密钥对

密钥对由私钥和公钥构成，是一组安全凭据，供您在连接到实例时证明自己的身份。

名称

名称最多可使用 255 个 ASCII 字符。它不能以空格开头或结尾。

文件格式

☒ pem
与 OpenSSH 共用

☐ ppk
与 PuTTY 共用

标签(可选)

此资源没有关联任何标签。

添加标签

您还可以另外添加 50 个标签

取消

创建密钥对

这样就会看到有新的密钥对了，接下来的所有需要密钥的地方我们都使用这一个密钥对就可以了。

• 创建 EC2 实例

接下来我们创建一个 EC2 实例。一个实例就是一个挂载在 AWS 上的虚拟机，我们通过远程连接可以接入并操作这个虚拟机，然后上面部署我们的 Web App。按理来说我们的前端和后端都需要部署，我是把他们部署到了两个不同的虚拟机上，创建方法是一样的，只是之后操作虚拟机的步骤不一样。在 **EC2** 的界面点击 **启动实例** 就会进入下面的页面，选择虚拟机的系统，我选择的是第二个。

The screenshot displays the AWS Management Console interface for creating an EC2 instance. The top navigation bar shows the user is logged in as 'vocstartsoft/user876174=jiang...' in the 'us-east-1' region. The left sidebar contains the 'EC2 Dashboard' and various navigation links. The main content area is titled '启动实例' (Launch Instance) and includes a '快速启动' (Quick Start) section. This section lists several AMIs (Amazon Machine Images) available for selection. The 'Amazon Linux 2018.03.0 (HVM), SSD Volume Type' is highlighted with a red box, indicating it is the selected option. Below this, there are sections for '快速启动' (Quick Start) and '快速启动' (Quick Start) with a list of AMIs, including Amazon Linux, Red Hat Enterprise Linux, SUSE Linux, and Ubuntu Server. The 'Amazon Linux 2018.03.0 (HVM), SSD Volume Type' is highlighted with a red box.

接着还要选实例类型，应该默认选的是标有免费的那个。一般来说后面可以用默认的配置，但是我还是简单自己配置一下，点击 **下一步**

aws 服务 资源组

1. 选择 AMI 2. 选择实例类型 3. 配置实例 4. 添加存储 5. 添加标签 6. 配置安全组 7. 审核

步骤 2: 选择一个实例类型

Amazon EC2 提供多种经过优化、适用于不同使用案例的实例类型以供选择。实例就是可以运行应用程序的虚拟服务器。它们由 CPU、内存、存储和网络容量组成不同的组合，可让您灵活地为您的应用程序选择适当的资源组合。有关实例类型以及这些类型如何满足您的计算需求的信息，请参阅[了解更多信息](#)。

筛选条件: 所有实例类型 最新一代 显示隐藏列

当前选择的实例类型: t2.micro (变量 ECU, 1 vCPU, 2.5 GHz, Intel Xeon Family, 1 GiB 内存, 仅限于 EBS)

	系列	类型	vCPU	内存 (GiB)	实例存储 (GiB)	可用的优化 EBS	网络性能	IPv6 支持
<input type="checkbox"/>	通用型	t2.nano	1	0.5	仅限于 EBS	-	低到中等	是
<input checked="" type="checkbox"/>	通用型	t2.micro	1	1	仅限于 EBS	-	低到中等	是
<input type="checkbox"/>	通用型	t2.small	1	2	仅限于 EBS	-	低到中等	是
<input type="checkbox"/>	通用型	t2.medium	2	4	仅限于 EBS	-	低到中等	是
<input type="checkbox"/>	通用型	t2.large	2	8	仅限于 EBS	-	低到中等	是
<input type="checkbox"/>	通用型	t2.xlarge	4	16	仅限于 EBS	-	中等	是
<input type="checkbox"/>	通用型	t2.2xlarge	8	32	仅限于 EBS	-	中等	是
<input type="checkbox"/>	通用型	t3a.nano	2	0.5	仅限于 EBS	是	高达 5Gb	是
<input type="checkbox"/>	通用型	t3a.micro	2	1	仅限于 EBS	是	高达 5Gb	是
<input type="checkbox"/>	通用型	t3a.small	2	2	仅限于 EBS	是	高达 5Gb	是
<input type="checkbox"/>	通用型	t3a.medium	2	4	仅限于 EBS	是	高达 5Gb	是
<input type="checkbox"/>	通用型	t3a.large	2	8	仅限于 EBS	是	高达 5Gb	是

取消 上一步 审核和启动 下一步: 配置实例详细信息

步骤3我没有改动它，直接点 下一步

aws 服务 资源组

1. 选择 AMI 2. 选择实例类型 3. 配置实例 4. 添加存储 5. 添加标签 6. 配置安全组 7. 审核

步骤 3: 配置实例详细信息

配置实例以便满足您的需求。您可以从同一 AMI 上启动多个实例。请求 Spot 实例以利用其低价优势，向实例分配访问管理角色等等。

实例的数量 1 启动至 Auto Scaling 组

购买选项 ☐ 请求 Spot 实例

网络 vpc-e080719d (默认) 新建 VPC

子网 无选项 (任何可用区的默认子网) 新建子网

自动分配公有 IP 使用子网设置 (启用)

置放群组 ☐ 将实例添加到置放群组

实例预置 打开

IAM 角色 无 创建新的 IAM 角色

关闭操作 停止

停止 - 休眠行为 ☐ 将休眠作为额外的停止行为启用

启用终止保护 ☐ 防止意外终止

监控 ☐ 启用 CloudWatch 详细监控 将收取额外费用。

租赁 共享 - 运行共享硬件实例 将对专用租赁收取额外的费用。

Elastic Inference ☐ 添加 Elastic Inference 加速器 将收取额外费用。

T2/T3 无限 ☐ 启用 可能收取额外费用

取消 上一步 审核和启动 下一步: 添加存储

这里添加储存，8G其实也没什么大问题好像，但是免费的额度可以有30G

aws 服务 资源组

1. 选择 AMI 2. 选择实例类型 3. 配置实例 4. 添加存储 5. 添加标签 6. 配置安全组 7. 审核

步骤 4: 添加存储

您的实例将使用以下存储设备设置启动。您可以将其他 EBS 卷和实例存储卷附加到您的实例，或编辑根卷的设置。您还可以在启动实例后附加其他 EBS 卷而非实例存储卷。详细了解 有关 Amazon EC2 中存储选项的信息。

卷类型 ①	设备 ①	快照 ①	大小 (GiB) ①	卷类型 ①	IOPS ①	吞吐量 (MB/s) ①	终止时删除 ①	加密 ①
根	/dev/xvda	snap-00e66c293c6f296fa	30	通用型 SSD (gp2)	100/3000	不适用	<input checked="" type="checkbox"/>	未加密

添加新卷

有资格使用免费套餐的客户最多可获得 30GB 的 EBS 通用型(SSD)或磁存储卷。有关免费使用套餐资格和使用限制的信息，请参阅“了解更多”。

取消 上一步 审核和启动 下一步: 添加标签

这里添加标签，给实例标一下名字，表示是服务器端。

aws 服务 资源组

1. 选择 AMI 2. 选择实例类型 3. 配置实例 4. 添加存储 5. 添加标签 6. 配置安全组 7. 审核

步骤 5: 添加标签

标签由一个区分大小写的键值对组成。例如，您可以定义一个键为 'Name' 且值为 'Webserver' 的标签。可将标签副本应用于卷和实例。标签将应用于所有实例和卷。有关标记 Amazon EC2 资源的信息，请参阅“了解更多”。

键 (最多 128 个字符)	值 (最多 256 个字符)	实例 ①	卷 ①
Name	WebServer	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

添加其他标签 (最多 50 个标签)

取消 上一步 审核和启动 下一步: 配置安全组

配置安全组，先可以像我这样弄，SSH的22端口是用来连接本机和实例，8080端口是访问我们的后端，80端口好像暂时没什么用。其他的还有 MongoDB 的 27017，MySQL 的 3306，Grafana 的 3000，Prometheus 的 9090 那些，都不用急着配，后面实例运行起来之后都是可以再改的，重点就是为了方便我把来源设置成了任何位置。

aws 服务 资源组 1. 选择 AMI 2. 选择实例类型 3. 配置实例 4. 添加存储 5. 添加标签 6. 配置安全组 7. 审核

步骤 6: 配置安全组

安全组是一组防火墙规则，用于控制您的实例的流量。在此页面上，您可以添加规则来允许特定流量到达您的实例。例如，如果您希望设置一个 Web 服务器，并允许 Internet 流量到达您的实例，请添加相应的规则来允许不受限制地访问 HTTP 和 HTTPS 端口。您可以创建一个新安全组或从下面选择一个现有安全组。有关 Amazon EC2 安全组的信息，请参阅“了解更多信息”。

分配安全组: ☒ 创建一个新的安全组 ☐ 选择一个现有的安全组

安全组名称:

描述:

类型 ①	协议 ①	端口范围 ①	来源 ①	描述 ①
SSH	TCP	22	任何位置 0.0.0.0/0::/0	例如 SSH for Admin Desktop
HTTP	TCP	80	任何位置 0.0.0.0/0::/0	例如 SSH for Admin Desktop
自定义 TCP 规则	TCP	8080	任何位置 0.0.0.0/0::/0	例如 SSH for Admin Desktop

添加规则

警告
设置为 0.0.0.0 的规则允许所有 IP 地址访问您的接口。我们建议将安全组规则设置为仅允许从已知的 IP 地址进行访问。

取消 上一步 **审核和启动**

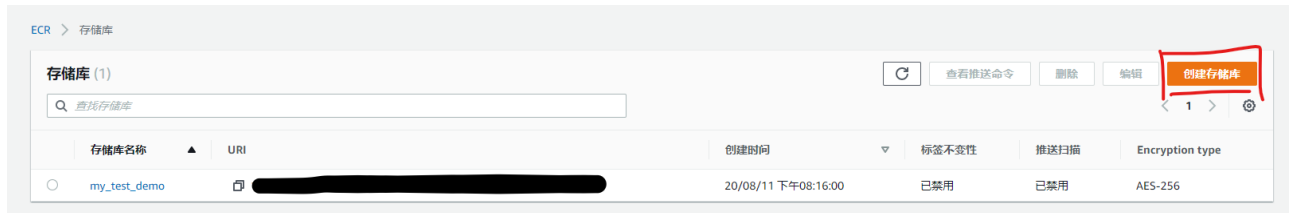
接着就可以点审核启动了，等待它审核启动，到 EC2 的控制面板去看，如果刚刚创建的实例的状态是下面这样就是可以用了。



关于实例连接的部分就可以看 **WSL** 相关的部分了。接下来介绍 **ECR**

- **使用 ECR 和推送镜像**

首先进入 ECR 的页面，选择新建储存库，然后填写储存库的名称就可以了。



创建存储库

存储库访问和标记

存储库名称

可以在存储库名称中包含命名空间(例如, namespace/repo-name)。

标签不变性
启用标签不变性, 以防止映像标签被使用相同标签的后续映像推送覆盖。禁用标签不变性, 以允许映像标签被覆盖。

☒ 已禁用

映像扫描设置

推送扫描
启用推送扫描, 以在将每个映像推送到存储库后对其进行自动扫描。如果已禁用, 则必须手动开始每个映像的扫描才能获取扫描结果。

☒ 已禁用

加密设置

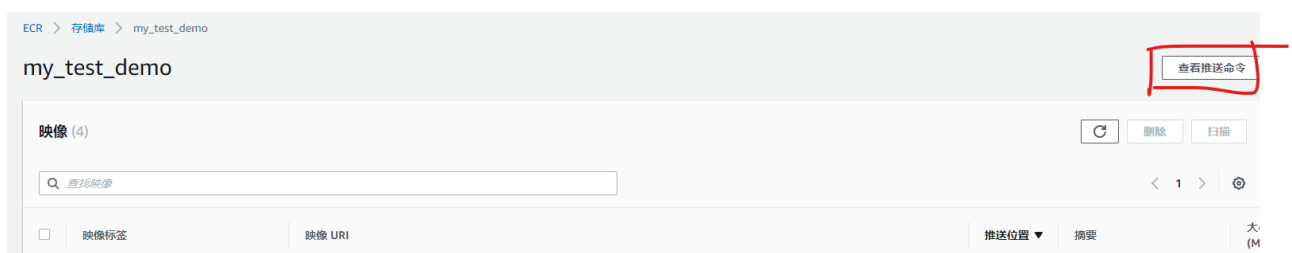
KMS 加密
您可以使用 AWS Key Management Service (KMS) 集成, 以便加密此存储库中存储的映像, 而不是使用默认加密设置。

☒ 已禁用

i The KMS encryption settings cannot be changed or disabled after the repository is created.

取消 创建存储库

接着会回到上一级页面, 点击储存库的名字进入储存库, 点击右上角的查看推送命令, 可以查看如何连接到储存库以及拉取、上传镜像。



在这里首先解释一下我们的项目的 Docker 的结构, 有五个容器, 容器内都是 Linux 的环境, 分别运行着 *后端*, *MySQL*, *MongoDB*, *Grafana*, *Prometheus*。后面四个的镜像都是从源地址 (Docker 的源可以换成阿里云) 拉取的, 后端的镜像就是 Springboot 中用 Maven 打包出来的 jar 包 (IDEA 里面橙黄色的那个目录)。

接下来要做的就是 Windows 系统上连接 ECR 然后把 jar 包推送到 ECR 中, 这样我们就可以在云端的服务器上拉取这个镜像。

首先确保 Windows 系统已经安装了 Docker (这个教程我还没有囊括了, 搜索的话搜 Docker Desktop 的教程就好, 只需要有 Docker 就可以了。注意所有涉及到 Docker 的命令都要在 Docker Desktop 在运行的前提下才能执行)


```
Credentials
```

```
AWS Access
Session started at: 2020-08-19T05:12:31-0700
Session to end   at: 2020-08-19T08:12:31-0700
Remaining session time: 2h37m21s
```

```
AWS Starter account
Term: 355 days 17:32:59
```

```
AWS CLI:
Copy and paste the following into ~/.aws/credentials
```

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUzfWh6oBq7Aab500K0W0k3yJlZn9gPvV419
region = us-east-1
source_profile = default
```

的推送命令

macOS / LinuxWindows

请确保您已安装 **AWS CLI** 和 **Docker** 的最新版本。有关更多信息，请参阅 [开始使用 Amazon ECR](#)。

使用以下步骤进行身份验证并将映像推送到存储库。有关其他注册表身份验证方法(包括 Amazon ECR 凭证助手)，请参阅 [注册表身份验证](#)。

- 检索身份验证令牌并向注册表验证 Docker 客户端身份。
使用 AWS CLI:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin
```

注意: 如果您在使用 AWS CLI 时收到错误，请确保您已安装 AWS CLI 和 Docker 的最新版本。
- 使用以下命令生成 Docker 映像。有关从头生成 Docker 文件的信息，请参阅说明 [此处](#)。如果您已生成映像，则可跳过此步骤:

```
docker build -t
```
- 生成完成后，标记您的映像，以便将映像推送到此存储库:

```
docker tag
```
- 运行以下命令将此映像推送到您新创建的 AWS 存储库:

```
docker push
```

关闭

把第一行命令直接复制到命令行中执行，应该就会返回 **Login Succeed** 说明已经把 Windows 和 ECR 连到一起了。（注意这个时候 Docker 是需要启动的）

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [版本 10.0.19041.450]
(c) 2020 Microsoft Corporation. 保留所有权利。

C:\Users\97471>aws ecr get-login-password --region us-east-1 | docker login --username AWS
Login Succeeded

C:\Users\97471>
```

接下来再 **项目目录** 下（就是有 Dockerfile 的那个目录下，也就是IDEA自带的控制台的默认路径下），执行第二个命令

```
docker build -t name:tag . //一定要注意最后有一个点 .
```

就可以把 jar 包 build 为 docker 的镜像了。（至于 jar 包打包请打开 **IDEA > View > Tool Window > Maven**, 选择里面的 package 并执行即可。）

接下来执行第三个命令，把自己刚刚命名的镜像标记为 ECR 给的这个名字（虽然我不知道有什么用）。最后执行第四个命令就可以把镜像推送到 ECR 的储存库里面了。可以在储存库里面找到就说明成功了，是可以拉取的镜像了。

PART.2 WSL

- **关于 WSL**

WSL 是 Windows Subsystem for Linux 的简称，相当于之前用的 VMware 虚拟机的感觉吧，但是体量更轻，更方便。具体的安装教程可以看：[WSL安装教程](#) 网上教程比较多，只要最后能够在 Microsoft Store 的应用市场搜索下载 **Ubuntu** 然后可以跑出来它的控制台界面就可以了（如下）。

PS：WSL默认安装子系统的位置是 C 盘，所以介意的话可以搜一下怎么改变子系统的存储路径。



- **开始使用 WSL 连接 EC2**

刚开始使用 WSL 需要安装一些必要的软件，一般来说都不要紧，遇到报错说命令不存在的时候再到 root 下 **apt-get**（有些是 **yum**）安装一下就好了。

为了后续的连接需要先准备密钥，先在 ~ 目录下面 **mkdir** 新建一个文件夹，然后 **cd** 到 **mnt** 目录下，**ls** 发现这个目录就是本机的硬盘的目录，找到之前保存的密钥文件，然后执行下面的命令把它复制到之前新建的文件夹里面去

```
xxx@user> cp xxx.pem ~/xxx
```

接下来回到那个文件夹，输入下面的命令给这个密钥文件权限（可能要在 root 下）

```
xxx@user> chmod 400 xxx.pem
```

然后就可以通过 SSH 连接到之前我们创建的实例虚拟机上了

```
xxx@user> ssh -i xxx.pem xxx@xxx.compute-1.amazonaws.com
```

这里的这些命令的具体值，可以在 EC2 的控制台的实例界面，点击左上角连接查看



注意这里官方给的命令里面，密钥文件加了双引号，但是我好像这样执行不了，是去掉双引号之后就可以的。

接下来 Ubuntu 的命令行界面就变成了操作 EC2 实例的命令行了，之后每一次连接实例的方法都是相同的，注意不同的实例是可以用一个密钥的，但是连接的时候实例本身的名字不一样。

PART.3 EC2

- **使用 EC2 实例**

理一下我们现在的处境，我们在 Windows 的一个 Ubuntu 子系统上面，远程连接了一个 EC2 的 Linux 实例，正在操作它的命令行。

接下来需要在这个 EC2 部署我们的 Docker 项目。

首先要设置 root 的密码，一般的实例默认是没有密码的，需要设置

```
ec2-user@xxx> sudo passwd root
```

接下来输入密码确认就可以了。

进入 root 模式后 `apt-get install` 或者 `yum install -y` 来下载 `docker` 和 `awscli`

注意需要

```
ec2-user@xxx> service docker start
```

来启动 Docker 服务

- **准备一些文件** 在 EC2 中新建一个文件夹 `app`，然后准备以下文件：**detail.json**, **ticket_demo.sql**, **docker-compose.yaml**, **prometheus.yml**。这些文件有两个方法，一是新建然后复制我们项目中对对应文件的内容（命令行复制可以用右键点光标），也可以用后面会讲的 **WinSCP** 直接传文件（这个更方便）。
- **EC2 连接 ECR**

在 EC2 上安装 AWS CLI 之后，`~` 目录下会出现一个 **.aws** 文件夹，可以用 `ls -al` 命令查看。里面同样的会有一个 `credentials` 文件，用之前同样的方法覆盖里面的认证信息，用之前同样的方法就可以连接到 ECR 了（只用 Login Succeed 就可以了）

接着输入下面的命令拉取之前我们上传的镜像，镜像的名字就是之前 push 的那个。

```
docker pull xxxxxxxx
```

这样接下来部署的时候就可以用了。

另外还有一种情况，是这个实例是自带 AWS CLI 的（就是安装的时候提示你已安装），它的目录好像就会不一样，所以可以用下面的命令设置认证信息

```
aws configure
```

接着填写对应的项目的内容，注意 region 那项填 **us-east-1**。此外还需要用

```
aws configure set aws_session_token xxxxxxxxxx
```

把最后一项的内容填进去，再尝试连接 ECR 就可以了。

- **EC2 部署 Docker** 请确保上面的步骤都完成了。

在 `app` 目录下（就是有 `docker-compose.yaml` 文件的那个目录）输入命令

```
docker-compose up
```

就可以自动完成部署的工作了。（如果提示没有这个命令，或许需要再安装一下 docker-compose，可以看官方的文档：[Docker-compose安装](#)）

这个时候应该会自动拉取镜像，部署容器，然后一大堆各色代码飞过，如果不想看见这些话请输入

```
docker-compose up -d
```

或者稳定后直接关闭 Ubuntu 的窗口（不是按 Ctrl+C！）在重新打开 Ubuntu 重新连接到 EC2 也可以继续操作。

- **Mysql 和 MongoDB**

新部署的 Docker 里面跑的数据库是没有数据的。

在 EC2 的命令行输入下面的命令

```
ec2-user@xxx> docker ps
```

可以查看容器状况，每一列有一条是容器的名字。输入

```
ec2-user@xxx> docker exec -it 容器的名字 bash
```

可以进入容器操作。如果报错没有 bash 什么的，可以试一下 bash 改成 sh

在 MySQL 的容器里面的命令行输入 `mysql -uroot -p` 进入 MySQL 密码是 root

然后创建 database `ticket_gathering`（反正要和你的 jar 包里面的后端里面的配置的名字一样），然后 use 这个 db，输入

```
source /opt/app/ticket_demo.sql;
```

就可以导入 MySQL 的数据了。

同样的方法进入 MongoDB 的容器，直接在容器的命令行输入

```
mongoimport --db ticket --collection ticketdetail --file  
/opt/app/detail.json
```

显示导入了 1335 条数据就可以了。这样后端应该就可以直接访问了。具体的 IP 地址在 EC2 实例的控制面板下面的信息里有。

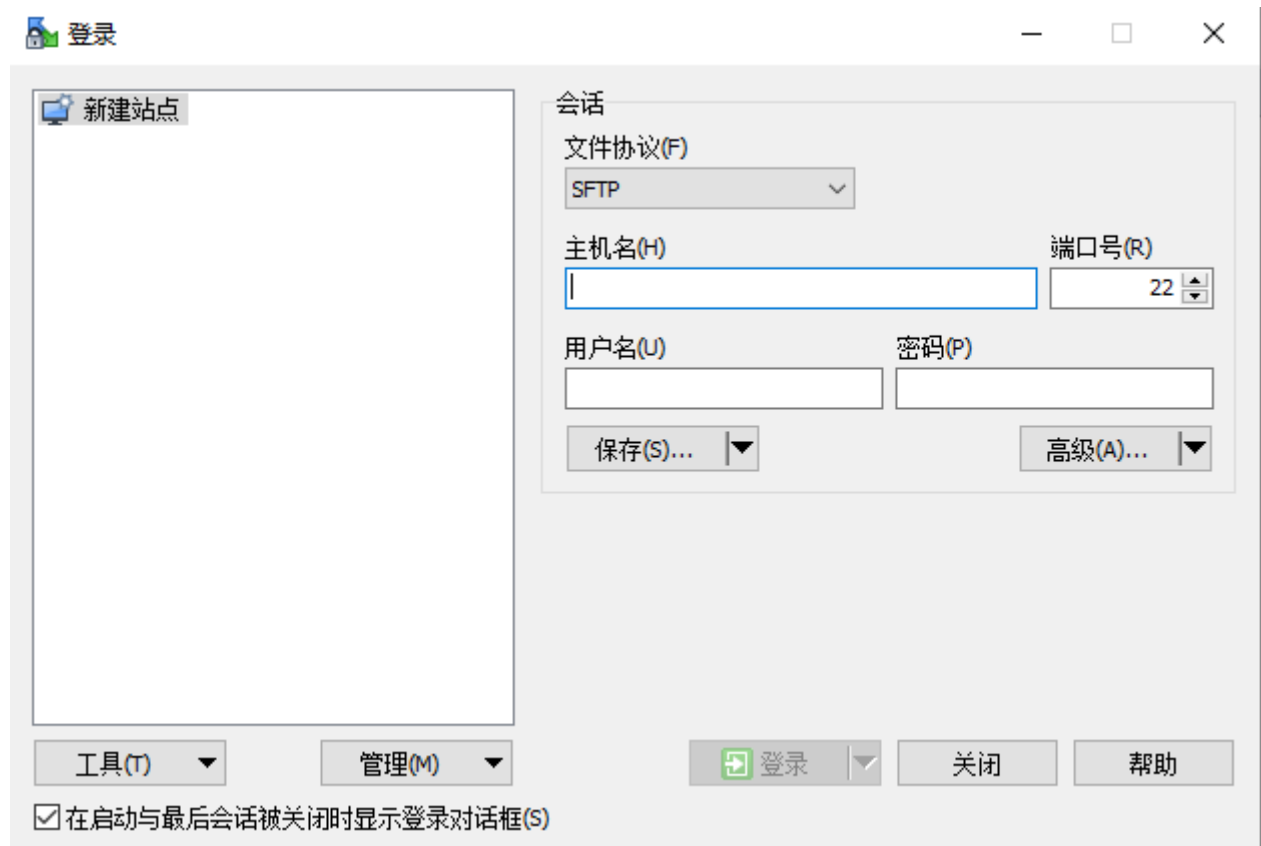
PART.4 WinSCP

- 关于 WinSCP

WinSCP 用来在本机和云端虚拟机之间传文件。

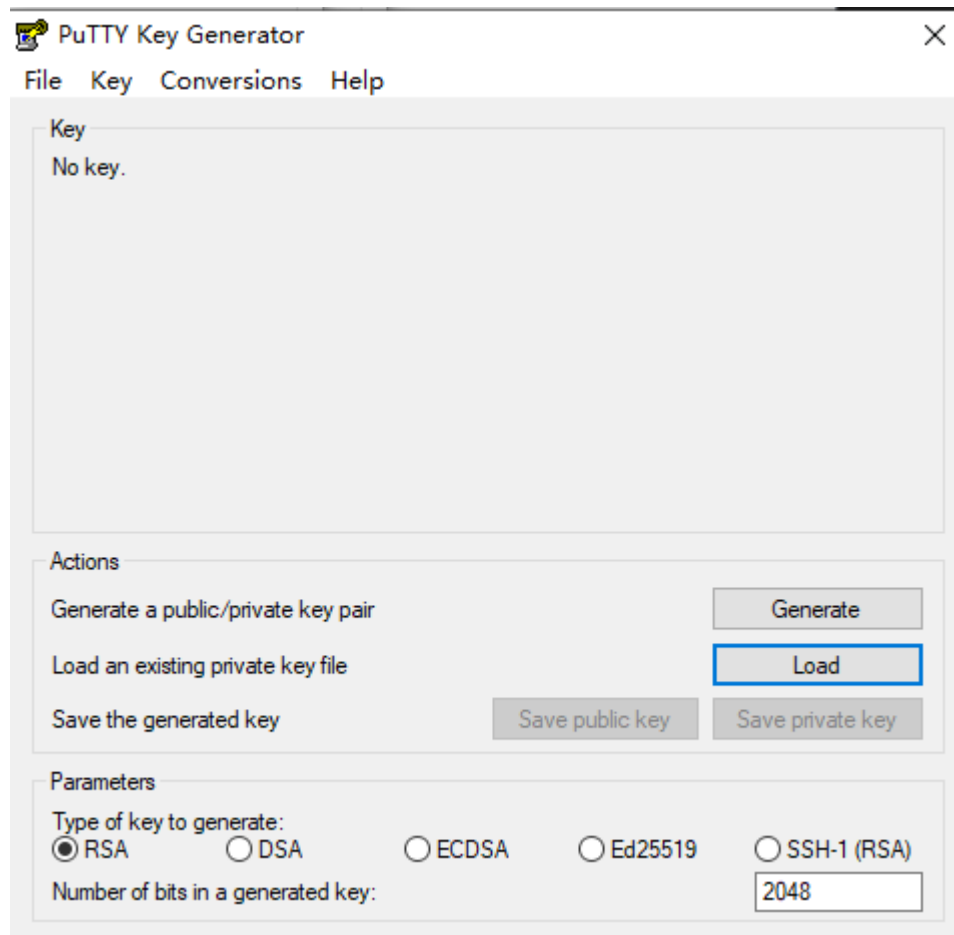
安装教程请百度

打开之后会弹出登录界面

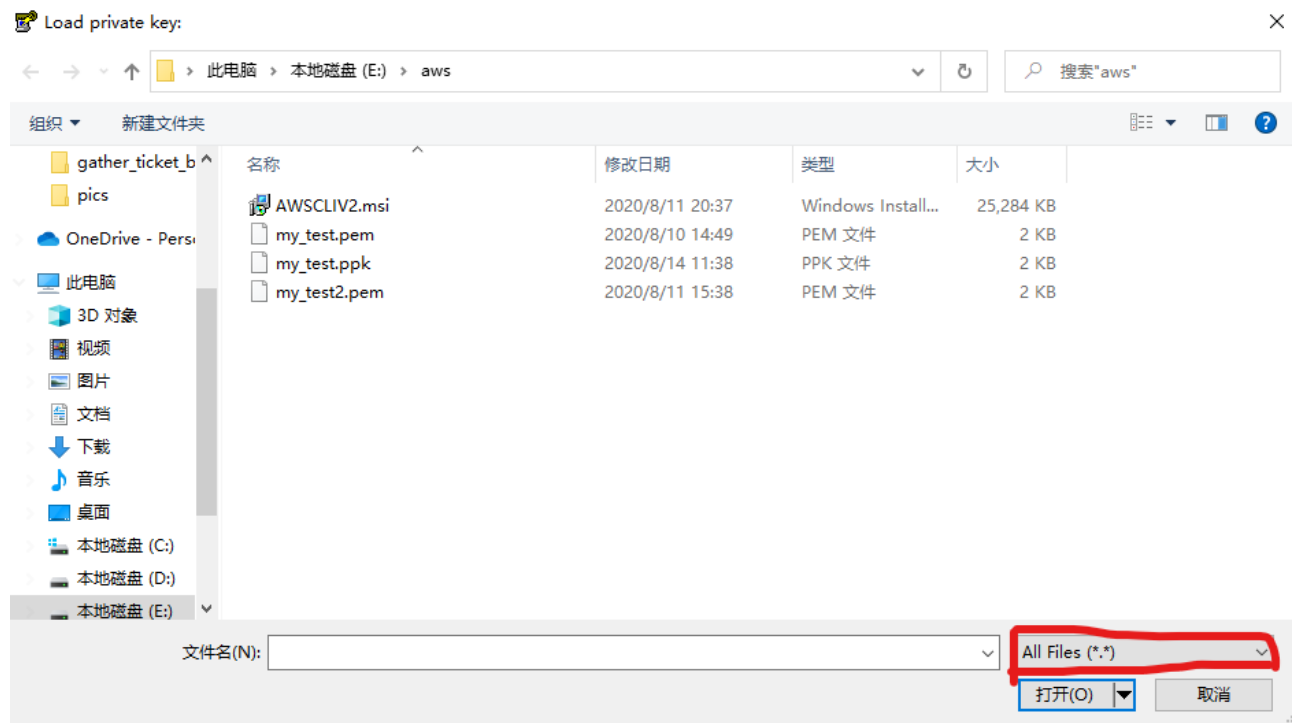


主机名填要连的那个 EC2 实例的**公有 IP 地址**，端口为 **22**，用户名为 **ec2-user**（如果之前的步骤都跟我一样的话）

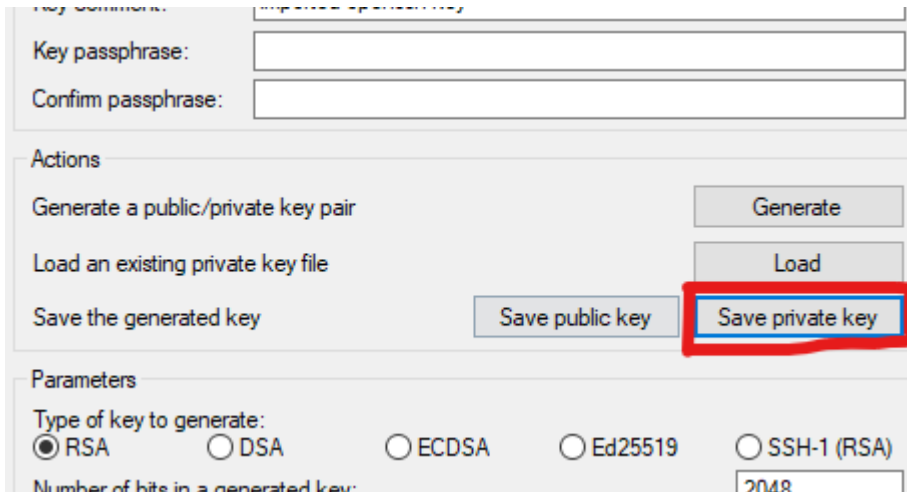
然后点 **高级>SSH>验证>验证参数>工具>使用PuTTYgen生成新的密钥对**



点击 **load**，找到之前保存在本地的密钥 .pem 文件（要在右下角把文件类型改成 All Files）



之后点击 **Save private key** 保存转换成的 .ppk 文件



The screenshot shows the AWS Key Management Service console. Under the 'Actions' section, there are three buttons: 'Generate', 'Load', and 'Save private key'. The 'Save private key' button is highlighted with a red box. Below the 'Actions' section, the 'Parameters' section shows the 'Type of key to generate' set to 'RSA' and the 'Number of bits in a generated key' set to '2048'.

最后再选择刚刚生成的 .ppk 文件，点击 **登录** 等待连接即可。接下来的操作包括新建文件夹、传送文件都很简单了。

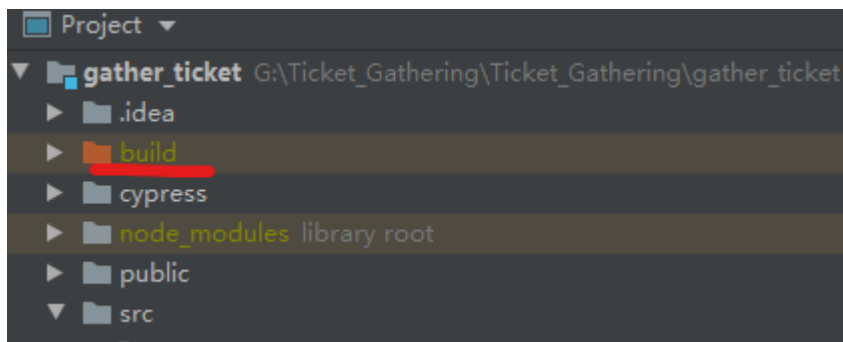
PART.5 Front End

- **前端代码打包**

首先在前端的代码目录下执行

```
npm run build
```

完了就会有一个 **build** 文件夹（橙黄色的），这个就是打包好的前端。



- **前端部署**

用上面同样的方法**新建一个 EC2 实例**，用来部署前端，具体的方法请翻回去看。

然后用 WinSCP 把刚刚打包的 build 文件夹整个复制到实例中的某个位置。

接着用 WSL 连接到实例之后先安装 Nginx

```
yum install -y nginx
```

安装好后启动 Nginx 服务

```
service nginx start
```

然后到它的根目录下面去修改默认的配置

```
cd /etc/nginx
vi nginx.conf
```

然后修改一下图中标注的位置，注意 root 的路径是自己刚刚保存 build 的路径。（在 root 模式下才能该这个文件）

```
ec2-user@ip-172-31-35-105:/etc/nginx
# For more information on configuration, see:
# * Official English Documentation: http://nginx.org/en/docs/
# * Official Russian Documentation: http://nginx.org/ru/docs/

user root;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /var/run/nginx.pid;

# Load dynamic modules. See /usr/share/doc/nginx/README.dynamic.
include /usr/share/nginx/modules/*.conf;

events {
    worker_connections 1024;
}

http {
    log_format main '$remote_addr - $remote_user [$time_local] "$request" '
        '$status $body_bytes_sent "$http_referer" '
        '"$http_user_agent" "$http_x_forwarded_for"';

    access_log /var/log/nginx/access.log main;

    sendfile        on;
    tcp_nopush      on;
    tcp_nodelay      on;
    keepalive_timeout 65;
    types_hash_max_size 2048;

    include /etc/nginx/mime.types;
    default_type application/octet-stream;

    # Load modular configuration files from the /etc/nginx/conf.d directory.
    # See http://nginx.org/en/docs/nginx_core_module.html#include
    # for more information.
    include /etc/nginx/conf.d/*.conf;

    index index.html index.htm;

    server {
        listen      80 default_server;
        listen      [::]:80 default_server;
        server_name localhost;
        root         /home/ec2-user/app/build;

        # Load configuration files for the default server block.
        include /etc/nginx/default.d/*.conf;

        location / {
            try_files $uri $uri/ /index.html;
        }

        # redirect server error pages to the static page /40x.html
        #
        error_page 404 /404.html;
        location = /40x.html {

```

修改好后重启 Nginx 服务

```
service nginx restart
```

之后就可以在这个部署了前端的实例的 IP 地址访问前端的页面了。

PS: 前端的代码里面我把 url 改成了一个常量在 src/Constant/constant.js , 如果你想自己部署一下后端, 需要把这个常量的值改成自己后端的 IP 地址。

至此我们的项目的前后端都已经部署完毕了, 应该没有什么问题是可以访问的了, 如果遇到什么问题就问我吧