Tutorial – AWS Infrastructure and Bookstack Set Up Using Scripts

Tamim Hemat
A01278451

In this tutorial I will walk you through the process of setting up an AWS VPC, EC2 Instance, and RDS MySQL Database along with a Bookstack instance through the EC2 Instance – all this using scripts.

First, I am going to list the components of the AWS Infrastructure:

- VPC
- Subnets
    - 1 public subnet for the EC2 Instance
    - 2 private subnets for our RDS database
- Internet Gateway
- Route table
- Security groups
    - 1 for the EC2 Instance, it will allow ssh and http traffic from anywhere
    - 1 for the RDS database, it will allow MySQL traffic from the VPC
- SSH key pair
- EC2 Instance
    - Ubuntu 22.04
- Subnet group
    - It will include the 2 private subnets and will be used for the database
- RDS database
    - MySQL

The infrastructure script will contain the commands for creating all these components. It will also include commands for updating the application script with some variables and transferring it to the EC2 instance and running it. It is **important** to note that before running the infrastructure script, the application script must be completed and reside in the same directory as this one.

Below I will show and explain all the commands for creating the AWS resources along with the corresponding “describe” commands:

First thing we need to do is create is a VPC and store it's ID in a variable.

```
VPC_ID=$(aws ec2 create-vpc \
--cidr-block 10.0.0.0/16 \
--tag-specification ResourceType=vpc,Tags=['{Key=Name,Value=as2-tamim-vpc}'] | yq
'.Vpc.VpcId')
```

All AWS-CLI commands start with the keyword "**aws**". For most of the infrastructure the following keyword is "**ec2**" for resources related to our VPC and EC2 instance.
After these keywords we use a specific command to create or modify a resource. In the example above we use the "**create-vpc**" command to create a VPC with the CIDR block "**10.0.0.0/16**" and a name of "**as2-tamim-vpc**". We then pipe the output of this command to the command "**yq**" which is a command that allows us to extract data from YAML input. We extract the "**VpcId**" property of the "**.Vpc**" element. The output of the whole command is stored in the variable "**VPC_ID**" that we will use in other commands. This pattern is repeated with almost all of our commands for creating resources, so I am explaining it in-depth now, but you'll notice it is pretty much the same with every other command. This pattern is useful because it allows us to create our resources while storing important metadata like IDs or names that we can use to create other resources.

Describe command for the VPC:

```
aws ec2 describe-vpcs --vpc-ids $VPC_ID
```

```
Vpcs:
- CidrBlock: 10.0.0.0/16
  CidrBlockAssociationSet:
  - AssociationId: vpc-cidr-assoc-02ddcf2f61342e3cd
    CidrBlock: 10.0.0.0/16
    CidrBlockState:
      State: associated
  DhcpOptionsId: dopt-01164ccd1cf085931
  InstanceTenancy: default
  IsDefault: false
  OwnerId: '741561355720'
  State: available
  Tags:
  - Key: Name
    Value: as2-tamim-vpc
  VpcId: vpc-0fb14483f7bcc4b8d
```

Next step is to create the 3 subnets that we need. We also store their IDs in variables.

```
SUBNET1=$(aws ec2 create-subnet \
--vpc-id $VPC_ID \
--cidr-block 10.0.1.0/24 \
--availability-zone us-west-2a \
--tag-specification ResourceType=subnet,Tags=['{Key=Name,Value=as2-rds-pub-ec2}']
| yq '.Subnet.SubnetId')

SUBNET2=$(aws ec2 create-subnet \
--vpc-id $VPC_ID \
--cidr-block 10.0.2.0/24 \
--availability-zone us-west-2a \
--tag-specification ResourceType=subnet,Tags=['{Key=Name,Value=as2-rds-pri-1}'] |
yq '.Subnet.SubnetId')

SUBNET3=$(aws ec2 create-subnet \
--vpc-id $VPC_ID \
--cidr-block 10.0.3.0/24 \
--availability-zone us-west-2b \
--tag-specification ResourceType=subnet,Tags=['{Key=Name,Value=as2-rds-pri-2}'] |
yq '.Subnet.SubnetId')
```

The command for all of them is "**create-subnet**". We just change the Availability Zone, CIDR block and name to the values that we desire. This command creates the subnets and again pipes the output to "**yq**" so we can extract and store the subnet IDs. It is **important** to note that all these subnets are created as **private** subnets. Our next command modifies the first subnet so it becomes a **public** one.

```
aws ec2 modify-subnet-attribute \
--subnet-id $SUBNET1 \
--map-public-ip-on-launch
```

This command enables the "**map-public-ip-on-launch**" property of the subnet to so it can assign IPv4 addresses to EC2 instances launched with it.

Describe command for the subnets:

```
aws ec2 describe-subnets --subnet-ids $SUBNET1 $SUBNET2 $SUBNET3
```

```
Subnets:
- AssignIpv6AddressOnCreation: false
  AvailabilityZone: us-west-2a
  AvailabilityZoneId: usw2-az1
  AvailableIpAddressCount: 251
  CidrBlock: 10.0.1.0/24
  DefaultForAz: false
  EnableDns64: false
  Ipv6CidrBlockAssociationSet: []
  Ipv6Native: false
  MapCustomerOwnedIpOnLaunch: false
  MapPublicIpOnLaunch: true
  OwnerId: '741561355720'
  PrivateDnsNameOptionsOnLaunch:
    EnableResourceNameDnsAAAARecord: false
    EnableResourceNameDnsARecord: false
    HostnameType: ip-name
  State: available
  SubnetArn: arn:aws:ec2:us-west-2:741561355720:subnet/subnet-03ce7975b96dec087
  SubnetId: subnet-03ce7975b96dec087
  Tags:
  - Key: Name
    Value: as2-rds-pub-ec2
  VpcId: vpc-0fb14483f7bcc4b8d
- AssignIpv6AddressOnCreation: false
  AvailabilityZone: us-west-2a
  AvailabilityZoneId: usw2-az1
  AvailableIpAddressCount: 251
  CidrBlock: 10.0.2.0/24
  DefaultForAz: false
  EnableDns64: false
  Ipv6CidrBlockAssociationSet: []
  Ipv6Native: false
  MapCustomerOwnedIpOnLaunch: false
  MapPublicIpOnLaunch: false
  OwnerId: '741561355720'
  PrivateDnsNameOptionsOnLaunch:
    EnableResourceNameDnsAAAARecord: false
    EnableResourceNameDnsARecord: false
    HostnameType: ip-name
  State: available
  SubnetArn: arn:aws:ec2:us-west-2:741561355720:subnet/subnet-0e8b459fbc4401204
  SubnetId: subnet-0e8b459fbc4401204
  Tags:
  - Key: Name
    Value: as2-rds-pri-1
  VpcId: vpc-0fb14483f7bcc4b8d
```

```
- AssignIpv6AddressOnCreation: false
  AvailabilityZone: us-west-2b
  AvailabilityZoneId: usw2-az2
  AvailableIpAddressCount: 251
  CidrBlock: 10.0.3.0/24
  DefaultForAz: false
  EnableDns64: false
  Ipv6CidrBlockAssociationSet: []
  Ipv6Native: false
  MapCustomerOwnedIpOnLaunch: false
  MapPublicIpOnLaunch: false
  OwnerId: '741561355720'
  PrivateDnsNameOptionsOnLaunch:
    EnableResourceNameDnsAAAARecord: false
    EnableResourceNameDnsARecord: false
    HostnameType: ip-name
  State: available
  SubnetArn: arn:aws:ec2:us-west-2:741561355720:subnet/subnet-05aa722b522bdda30
  SubnetId: subnet-05aa722b522bdda30
  Tags:
  - Key: Name
    Value: as2-rds-pri-2
  VpcId: vpc-0fb14483f7bcc4b8d
```

Next step is to create an Internet Gateway and attach it to our VPC. We need 2 commands to achieve this:

```
IGW_ID=$(aws ec2 create-internet-gateway \
--tag-specification ResourceType=internet-gateway,Tags=['{Key=Name,Value=as2-igw}'] | yq '.InternetGateway.InternetGatewayId')

aws ec2 attach-internet-gateway \
--internet-gateway-id $IGW_ID \
--vpc-id $VPC_ID >/dev/null
```

The first command is "**create-internet-gateway**" where we only need to specify the resource type and name, and pipe to store the ID. The second command we use is "**attach-internet-gateway**" where we specify our newly created IGW ID and the ID of the VPC we want to attach it too.

Describe command for our IGW:

```
aws ec2 describe-internet-gateways --internet-gateway-ids $IGW_ID
```

```
InternetGateways:
- Attachments:
  - State: available
    VpcId: vpc-0fb14483f7bcc4b8d
  InternetGatewayId: igw-08992d19e48bbeb27
  OwnerId: '741561355720'
  Tags:
  - Key: Name
    Value: as2-igw
```

Next step is to create a route table, create a route for our IGW in the route table, and associate our public subnet with the route table. We will use 3 commands for that. First:

```
ROUTE_TABLE_ID=$(aws ec2 create-route-table \
--vpc-id $VPC_ID \
--tag-specification ResourceType=route-table,Tags=['{Key=Name,Value=as2-rt}'] |
yq '.RouteTable.RouteTableId')
```

We use the command "**create-route-table**" and provide the VPC ID. We store the route table ID in a variable. After that:

```
aws ec2 create-route \
--route-table-id $ROUTE_TABLE_ID \
--destination-cidr-block 0.0.0.0/0 \
--gateway-id $IGW_ID >/dev/null
```

We use the command "**create-route**" to create a route with a CIDR block of 0.0.0.0/0 (i.e. anywhere) for our IGW. Since with this command we don't store anything in a variable it is going to produce some output on the command line. To avoid this, we pipe this output to /dev/null with "**>/dev/null**". The last command is:

```
aws ec2 associate-route-table \
--subnet-id $SUBNET1 \
--route-table-id $ROUTE_TABLE_ID >/dev/null
```

We use the command "**associate-route-table**" to associate our public subnet with our route table. Here again we pipe the output to /dev/null since we don't store anything.

Describe command for the route table:

```
aws ec2 describe-route-tables --route-table-ids $ROUTE_TABLE_ID
RouteTables:
- Associations:
  - AssociationState:
      State: associated
    Main: false
    RouteTableAssociationId: rtbassoc-0a55715fb041d0485
    RouteTableId: rtb-01d7318201db12aa4
    SubnetId: subnet-03ce7975b96dec087
  OwnerId: '741561355720'
  PropagatingVgws: []
  RouteTableId: rtb-01d7318201db12aa4
  Routes:
  - DestinationCidrBlock: 10.0.0.0/16
    GatewayId: local
    Origin: CreateRouteTable
    State: active
  - DestinationCidrBlock: 0.0.0.0/0
    GatewayId: igw-08992d19e48bbeb27
    Origin: CreateRoute
    State: active
  Tags:
  - Key: Name
    Value: as2-rt
  VpcId: vpc-0fb14483f7bcc4b8d
```

Next step is to create a security group for our EC2 instance and give it some inbound rules. Here are the commands for that:

```
SECURITY_GROUP_ID=$(aws ec2 create-security-group \
--group-name rds-ec2-sg \
--description "Security group for EC2 in Assignment 2. Allows SSH and HTTP from
anywhere." \
--vpc-id $VPC_ID | yq '.GroupId')
```

We use the command "**create-security-group**" providing a name, description and our VPC ID. We store the ID of the security group in a variable. The next commands are:

```
aws ec2 authorize-security-group-ingress \
--group-id $SECURITY_GROUP_ID \
--protocol tcp --port 22 \
--cidr 0.0.0.0/0 >/dev/null

aws ec2 authorize-security-group-ingress \
--group-id $SECURITY_GROUP_ID \
--protocol tcp --port 80 \
--cidr 0.0.0.0/0 >/dev/null
```

The command we use here is "**authorize-security-group-ingress**". This creates security rules for a security group. We need to provide the security group ID along with the protocol, port, and allowed CIDR block for the rules. Here we create inbound security rules that allow SSH and HTTP traffic from anywhere. Once again – we don't store anything and we don't want cluttered command line, so we pipe to /dev/null.

Describe command for the EC2 security group:

```
aws ec2 describe-security-groups --group-ids $SECURITY_GROUP_ID
SecurityGroups:
- Description: Security group for EC2 in Assignment 2. Allows SSH from anywhere.
  GroupId: sg-04a35a753a807eb1c
  GroupName: rds-ec2-sg
  IpPermissions:
  - FromPort: 80
    IpProtocol: tcp
    IpRanges:
    - CidrIp: 0.0.0.0/0
    Ipv6Ranges: []
    PrefixListIds: []
    ToPort: 80
    UserIdGroupPairs: []
  - FromPort: 22
    IpProtocol: tcp
    IpRanges:
    - CidrIp: 0.0.0.0/0
    Ipv6Ranges: []
    PrefixListIds: []
    ToPort: 22
    UserIdGroupPairs: []
  IpPermissionsEgress:
  - IpProtocol: '-1'
    IpRanges:
    - CidrIp: 0.0.0.0/0
    Ipv6Ranges: []
    PrefixListIds: []
    UserIdGroupPairs: []
  OwnerId: '741561355720'
  VpcId: vpc-0fb14483f7bcc4b8d
```

Next step is to create a key pair for our EC2 instance.

```
aws ec2 create-key-pair \
--key-name as2-ec2-key \
--query 'KeyMaterial' \
--output text > as2-key.pem
```

The command we use is "**create-key-pair**". We only need to give our key a name. The **--query 'KeyMaterial'** extracts the key and the **--output text** converts it to text that we can store. We do this by redirecting to a new file that I here named "**as2-key.pem**". Optionally we can also specify the type of the key. The default is RSA but we can specify ED25519 by adding the parameter "**--key-type ed25519**" in the command.

We need to change the permissions of our key so it's not too open. We do this using the "**chmod**" command. A good practice is leaving read permissions for the root user only:

```
chmod 400 as2-key.pem
```

Describe command for the key pair:

```
aws ec2 describe-key-pairs --key-names as2-ec2-key
KeyPairs:
- CreateTime: '2023-03-04T04:08:28.369000+00:00'
  KeyFingerprint: 0b:af:52:b2:dc:ae:65:5f:37:41:5d:99:8d:a8:00:60:a5:5f:8f:d8
  KeyName: as2-ec2-key
  KeyPairId: key-04f562a8fbc08734e
  KeyType: rsa
  Tags: []
```

Next step is launching our EC2 instance:

```
EC2_ID=$(aws ec2 run-instances \
--image-id ami-0735c191cf914754d \
--count 1 \
--instance-type t2.micro \
--key-name as2-ec2-key \
--security-group-ids $SECURITY_GROUP_ID \
--subnet-id $SUBNET1 \
--associate-public-ip-address \
--tag-specification ResourceType=instance,Tags=['{Key=Name,Value=as2-rds-ec2}'] |
yq '.Instances[0].InstanceId')
```

We use the command "**run-instances**" to launch an EC2 instance. Here we need to specify the AMI image-id, instance type, count of instances (1 for this tutorial), security group, key pair, and subnet. We also specify the name of our instance and we also need to include the parameter "**--associate-public-ip-address**" so that our instance receives an IPv4 public address from the subnet. We don't need to specify our VPC or Availability Zone because the instance inherits them from the subnet it's launched in.

```
aws ec2 wait instance-running \
--instance-ids $EC2_ID
```

Using the stored EC2 instance ID we can use the "**wait instance-running**" command to ensure that our script won't continue until the EC2 instance is launched and fully running. We implement this command to ensure that our script won't reach another command that needs a variable from the EC2 instance metadata that is still not accessible. This is because launching an instance takes longer that most of the other commands. For example, creating a VPC happens in about 1-2 seconds while launching an EC2 instance can take between 1-2 minutes. It is good practice to implement "**wait**" commands when creating resources that take longer to start.

Describe command for the EC2 instance:

```
aws ec2 describe-instances --instance-ids $EC2_ID
```


```
Reservations:
- Groups: []
  Instances:
  - AmiLaunchIndex: 0
    Architecture: x86_64
    BlockDeviceMappings:
    - DeviceName: /dev/sda1
      Ebs:
        AttachTime: '2023-03-04T04:08:31+00:00'
        DeleteOnTermination: true
        Status: attached
        VolumeId: vol-09beb125a5087c0a5
    CapacityReservationSpecification:
      CapacityReservationPreference: open
    ClientToken: 6f0c8bc0-dcda-4ede-a383-0a8ff4568dc1
    CpuOptions:
      CoreCount: 1
      ThreadsPerCore: 1
    EbsOptimized: false
    EnaSupport: true
    EnclaveOptions:
      Enabled: false
    HibernationOptions:
      Configured: false
    Hypervisor: xen
    ImageId: ami-0735c191cf914754d
    InstanceId: i-0e7febed055738da8
    InstanceType: t2.micro
    KeyName: as2-ec2-key
    LaunchTime: '2023-03-04T04:08:31+00:00'
```

```
    MaintenanceOptions:
      AutoRecovery: default
    MetadataOptions:
      HttpEndpoint: enabled
      HttpProtocolIpv6: disabled
      HttpPutResponseHopLimit: 1
      HttpTokens: optional
      InstanceMetadataTags: disabled
      State: applied
    Monitoring:
      State: disabled
    NetworkInterfaces:
    - Association:
        IpOwnerId: amazon
        PublicDnsName: ''
        PublicIp: 35.93.31.175
      Attachment:
        AttachTime: '2023-03-04T04:08:31+00:00'
        AttachmentId: eni-attach-0146721908a6ae77c
        DeleteOnTermination: true
        DeviceIndex: 0
        NetworkCardIndex: 0
        Status: attached
      Description: ''
      Groups:
      - GroupId: sg-04a35a753a807eb1c
        GroupName: rds-ec2-sg
      InterfaceType: interface
      Ipv6Addresses: []
      MacAddress: 02:48:77:a1:98:bb
```

```
      NetworkInterfaceId: eni-076fc69822764a141
      OwnerId: '741561355720'
      PrivateIpAddress: 10.0.1.35
      PrivateIpAddresses:
      - Association:
          IpOwnerId: amazon
          PublicDnsName: ''
          PublicIp: 35.93.31.175
        Primary: true
        PrivateIpAddress: 10.0.1.35
      SourceDestCheck: true
      Status: in-use
      SubnetId: subnet-03ce7975b96dec087
      VpcId: vpc-0fb14483f7bcc4b8d
    Placement:
      AvailabilityZone: us-west-2a
      GroupName: ''
      Tenancy: default
    PlatformDetails: Linux/UNIX
    PrivateDnsName: ip-10-0-1-35.us-west-2.compute.internal
    PrivateDnsNameOptions:
      EnableResourceNameDnsAAAARecord: false
      EnableResourceNameDnsARecord: false
      HostnameType: ip-name
    PrivateIpAddress: 10.0.1.35
    ProductCodes: []
    PublicDnsName: ''
    PublicIpAddress: 35.93.31.175
    RootDeviceName: /dev/sda1
    RootDeviceType: ebs
    SecurityGroups:
    - GroupId: sg-04a35a753a807eb1c
      GroupName: rds-ec2-sg
    SourceDestCheck: true
    State:
      Code: 16
      Name: running
    StateTransitionReason: ''
    SubnetId: subnet-03ce7975b96dec087
    Tags:
    - Key: Name
      Value: as2-rds-ec2
    UsageOperation: RunInstances
    UsageOperationUpdateTime: '2023-03-04T04:08:31+00:00'
    VirtualizationType: hvm
    VpcId: vpc-0fb14483f7bcc4b8d
  OwnerId: '741561355720'
  ReservationId: r-06ea4281ae443a93c
```

Next step is creating a subnet group for our RDS database. We must choose 2 private subnets, but 1 of them must be in the same availability zone as our public subnet.

```
RDS_SUBNET_GROUP_NAME=$(aws rds create-db-subnet-group \
--db-subnet-group-name as2-rds-subnet-group \
--db-subnet-group-description "Subnet group for RDS in Assignment 2." \
--subnet-ids $SUBNET2 $SUBNET3 | yq '.DBSubnetGroup.DBSubnetGroupName')
```

For commands related to RDS we don't use the "**ec2**" keywords after "**aws**", but we use "**rds**". For creating the subnet group we use the command "**create-db-subnet-group**". We provide a name and a description for the subnet group and we also provide the IDs of our private subnets that we created earlier. We store the name of the subnet group.

Describe command for the DB subnet group:

```
aws rds describe-db-subnet-groups --db-subnet-group-name $RDS_SUBNET_GROUP_NAME
DBSubnetGroups:
- DBSubnetGroupArn: arn:aws:rds:us-west-2:741561355720:subgrp:as2-rds-subnet-group
  DBSubnetGroupDescription: Subnet group for RDS in Assignment 2.
  DBSubnetGroupName: as2-rds-subnet-group
  SubnetGroupStatus: Complete
  Subnets:
  - SubnetAvailabilityZone:
      Name: us-west-2a
    SubnetIdentifier: subnet-0e8b459fbc4401204
    SubnetOutpost: {}
    SubnetStatus: Active
  - SubnetAvailabilityZone:
      Name: us-west-2b
    SubnetIdentifier: subnet-05aa722b522bdda30
    SubnetOutpost: {}
    SubnetStatus: Active
  SupportedNetworkTypes:
  - IPV4
  VpcId: vpc-0fb14483f7bcc4b8d
```

Next step is creating a security group for our RDS database and adding an inbound security rule. The commands are very similar to the ones for the EC2 security group:

```
RDS_SECURITY_GROUP_ID=$(aws ec2 create-security-group \
--group-name rds-sg \
--description "Security group for RDS in Assignment 2. Allows MySQL access from
the VPC only." \
--vpc-id $VPC_ID | yq '.GroupId')

aws ec2 authorize-security-group-ingress \
--group-id $RDS_SECURITY_GROUP_ID \
--protocol tcp --port 3306 \
--cidr 10.0.0.0/16 >/dev/null
```

We create the security group in our VPC and store the ID in a variable. Then we add an inbound rule that allows MySQL traffic only from our VPC CIDR. Make sure that you specify the same CIDR block as the one used for creating the VPC. Alternatively, you can create a CIDR variable beforehand and use that instead.

Describe command for our RDS database security group:

```
aws ec2 describe-security-groups --group-ids $RDS_SECURITY_GROUP_ID
SecurityGroups:
- Description: Security group for RDS in Assignment 2. Allows MySQL access from the
    VPC only.
  GroupId: sg-09ffd220b396c1c0b
  GroupName: rds-sg
  IpPermissions:
  - FromPort: 3306
    IpProtocol: tcp
    IpRanges:
    - CidrIp: 10.0.0.0/16
    Ipv6Ranges: []
    PrefixListIds: []
    ToPort: 3306
    UserIdGroupPairs: []
  IpPermissionsEgress:
  - IpProtocol: '-1'
    IpRanges:
    - CidrIp: 0.0.0.0/0
    Ipv6Ranges: []
    PrefixListIds: []
    UserIdGroupPairs: []
  OwnerId: '741561355720'
  VpcId: vpc-0fb14483f7bcc4b8d
```

Next and last step for creating our AWS resources is creating our RDS database.

```
RDS_ID=$(aws rds create-db-instance \
--db-name as2_rds \
--db-instance-identifier as2-rds \
--db-instance-class db.t3.micro \
--engine mysql \
--master-username "$DB_MASTER_USER" \
--master-user-password "$DB_MASTER_PASS" \
--allocated-storage 20 \
--vpc-security-group-ids $RDS_SECURITY_GROUP_ID \
--db-subnet-group-name $RDS_SUBNET_GROUP_NAME \
--no-publicly-accessible \
--engine-version 8.0.28 \
--storage-type gp2 \
--availability-zone us-west-2a | yq '.DBInstance.DBInstanceIdentifier')
```

We use the command "**create-db-instance**" and the parameters we need to specify are db-name, db-instance-identified (ID), db-instance-class (equivalent to instance-type in EC2), engine, master username and password, allocated-storage, security groups and subnet group, and availability zone. For our username and password I use variables that I have defined at the beginning of the script. I used sample values, you can change them to whatever you want:

```
### Define the database master username and password ###

DB_MASTER_USER="admin"
DB_MASTER_PASS="adminpass"
```

For the db-instance-class we want to use "**db.t3.micro**" or "**db.t2.micro**" because they are free-tier classes. For engine we choose "**mysql**" because our Bookstack instance will use a MySQL database. For security group and subnet group we use the variables that we stored from creating them with the previous commands. We don't want our database to publicly accessible, so we define the parameter "**no-publicly-accessible**". The engine-version and storage-type are optional parameters, but I included them for demonstration purposes. The storage-type I chose here – "**gp2**" is the default one that comes with the **db.t3.micro** instance class. After all parameters are set, we can run the command and store the database ID in a variable.
Since this is the command that will take the longest to run, because creating and backing up the database is a slow process, we can implement the "**wait**" command again so that our database is up and running before the script continues execution:

```
aws rds wait db-instance-available \
--db-instance-identifier $RDS_ID
```

The "**aws rds wait db-instance-available**" command provides this functionality and "freezes" the script until the database has been completely set up.

Describe command for the RDS database:

```
aws rds describe-db-instances --db-instance-identifier $RDS_ID
DBInstances:
- ActivityStreamStatus: stopped
  AllocatedStorage: 20
  AssociatedRoles: []
  AutoMinorVersionUpgrade: true
  AvailabilityZone: us-west-2a
  BackupRetentionPeriod: 1
  BackupTarget: region
  CACertificateIdentifier: rds-ca-2019
  CertificateDetails:
    CAIdentifier: rds-ca-2019
    ValidTill: '2024-08-22T17:08:50+00:00'
  CopyTagsToSnapshot: false
```

```
CustomerOwnedIpEnabled: false
DBInstanceArn: arn:aws:rds:us-west-2:741561355720:db:as2-rds
DBInstanceClass: db.t3.micro
DBInstanceIdentifier: as2-rds
DBInstanceStatus: available
DBName: as2_rds
DBParameterGroups:
- DBParameterGroupName: default.mysql8.0
  ParameterApplyStatus: in-sync
DBSecurityGroups: []
DBSubnetGroup:
  DBSubnetGroupDescription: Subnet group for RDS in Assignment 2.
  DBSubnetGroupName: as2-rds-subnet-group
  SubnetGroupStatus: Complete
  Subnets:
  - SubnetAvailabilityZone:
      Name: us-west-2a
    SubnetIdentifier: subnet-0e8b459fbc4401204
    SubnetOutpost: {}
    SubnetStatus: Active
  - SubnetAvailabilityZone:
      Name: us-west-2b
    SubnetIdentifier: subnet-05aa722b522bdda30
    SubnetOutpost: {}
    SubnetStatus: Active
  VpcId: vpc-0fb14483f7bcc4b8d
DbInstancePort: 0
DbiResourceId: db-MQCBFPMOGWUHBPIC2E7PAPU4IY
DeletionProtection: false
DomainMemberships: []
```

```
  Endpoint:
    Address: as2-rds.c5dqoalqa93e.us-west-2.rds.amazonaws.com
    HostedZoneId: Z1PVIF0B656C1W
    Port: 3306
  Engine: mysql
  EngineVersion: 8.0.28
  IAMDatabaseAuthenticationEnabled: false
  InstanceCreateTime: '2023-03-04T04:13:37.789000+00:00'
  LatestRestorableTime: '2023-03-04T04:15:03.817000+00:00'
  LicenseModel: general-public-license
  MasterUsername: admin
  MonitoringInterval: 0
  MultiAZ: false
  NetworkType: IPV4
  OptionGroupMemberships:
  - OptionGroupName: default:mysql-8-0
    Status: in-sync
  PendingModifiedValues: {}
  PerformanceInsightsEnabled: false
  PreferredBackupWindow: 12:59-13:29
  PreferredMaintenanceWindow: mon:11:28-mon:11:58
  PubliclyAccessible: false
  ReadReplicaDBInstanceIdentifiers: []
  StorageEncrypted: false
  StorageThroughput: 0
  StorageType: gp2
  TagList: []
  VpcSecurityGroups:
  - Status: active
    VpcSecurityGroupId: sg-09ffd220b396c1c0b
```


After the database has been set up, we are finished with setting up our AWS infrastructure and all resources have been created.

The next part of the infrastructure script has the following functionality:

- Update the DB username, password and endpoint variables in the application script
- Get the public IP of the newly created EC2 instance
- Transferring the application script to the EC2 instance
- Running the application script from inside the EC2 instance

We will leave this part for now and we're going to shift to the **application script**. The reasoning is that we need to have the script completed before we run the infrastructure script and so we can understand the command in our infrastructure script that changes a part of the application script. After explaining the application script we'll come back to the last steps of the infrastructure script.

APPLICATION SCRIPT

Here are the steps needed to setup Bookstack with our application script:

1. Install required packages
2. Create the MySQL database that will be used by Bookstack
3. Download Bookstack by cloning from Git
4. Install Composer (a PHP dependency manager)
5. Install required PHP dependencies with Composer
6. Create the .env file for our Bookstack with the correct variables
7. Run Bookstack database migration
8. Change file and folder permissions for Bookstack
9. Configure the Nginx server

All these steps are achieved by functions. Each step is a separate function. Apart from these functions, we have a few helper functions, since our script is intended for a fresh Ubuntu environment (i.e. no previous MySQL or Nginx server).

Our script starts by defining some variables

```
#!/bin/bash

echo "This script installs a new BookStack instance on a fresh Ubuntu 22.04
server."
echo "This script does not ensure system security."
echo ""

LOGPATH=$(realpath "bookstack_install_$(date +%s).log")

SCRIPT_USER="${SUDO_USER:-$USER}"

DB_PASS="$(head /dev/urandom | tr -dc A-Za-z0-9 | head -c 13)"

BOOKSTACK_DIR="/var/www/bookstack"

INSTANCE_IP=$(curl -s https://checkip.amazonaws.com)

DOMAIN=$1
# Set the RDS master username, password and endpoint
```

```
# These will be automaticall substituted to the correct values in the
infrastructure script
DB_MASTER_USER=EnterUsernameHere
DB_MASTER_PASS=EnterPasswordHere
RDS_ENDPOINT=EnterEndpointHere

export DEBIAN_FRONTEND=noninteractive
```

The last 3 variables are just placeholders. They will be replaced by the infrastructure script, but I will explain that when we return to it.

The variable "**DOMAIN=$1** is the command line argument provided when the application script is run. It could either be a domain name or an IP address. In this tutorial we use the IP of the EC2 instance, but we'll get to that later.
The line "**export DEBIAN_FRONTEND=noninteractive**" sets the global environment variable so that we don't get interactive prompts during our application setup.

The rest of the variables are self-explanatory.

The first helper function we have is the following:

```
function error_out() {
  echo "ERROR: $1" | tee -a "$LOGPATH" 1>&2
  exit 1
}
```

This function prints out an error message passed as an argument on the command line and to our log file if an error is encountered. Gladly for us, this script runs without any of these :D.

The next helper function is:

```
function info_msg() {
  echo "$1" | tee -a "$LOGPATH"
}
```

This function takes an argument that is in the form of an informative message and prints it out to the command line and our log file.

The next helper function is:

```
function run_pre_install_checks() {
  if [[ $EUID -gt 0 ]]
  then
    error_out "This script must be ran with root/sudo privileges"
  fi
```

```
  if [ -d "/etc/nginx/sites-enabled" ]
  then
    error_out "This script is intended for a fresh server install, existing nginx
config found, aborting install"
  fi

  if [ -d "/var/lib/mysql" ]
  then
    error_out "This script is intended for a fresh server install, existing MySQL
data found, aborting install"
  fi
}
```

This is more of a checker function. The first if statement checks if the script is run with **sudo** privileges (i.e. sudo on the command line when running the script). This is needed as there are some commands requiring root privileges. If not ran as root, the script exits and gives an error message.
The second if statement checks if the **/etc/nginx/sites-enabled** directory exists. Since this script is intended for a fresh Ubuntu system if this directory exists that means Nginx has already been installed. In that case the script exits with an error message.
The last if statement checks if the **/var/lib/mysql** directory exists. If it does then this is not a fresh Ubuntu system. In that case the script exists with an error message.

Our last helper function is:

```
function run_prompt_for_domain_if_required() {
  if [ -z "$DOMAIN" ]
  then
    info_msg ""
    info_msg "Enter the domain (or IP if not using a domain) you want to host
BookStack on and press [ENTER]."
    info_msg "Examples: my-site.com or docs.my-site.com or $(curl -s
https://checkip.amazonaws.com)"
    read -r DOMAIN
  fi

  # Error out if no domain was provided
  if [ -z "$DOMAIN" ]
  then
    error_out "A domain must be provided to run this script"
  fi
}
```

This function checks if a domain/IP has been provided as an argument to the script. If not the script informs about that and tried to read for input from the command line.

If non is provided again, the script exits with an error message.

Next, I am going to describe the main functions of the script:

1. Install required packages

```
function run_package_installs() {
  apt update
  apt install -y git unzip nginx php8.1 curl php8.1-curl php8.1-mbstring php8.1-
ldap \
  php8.1-xml php8.1-zip php8.1-gd php8.1-mysql php8.1-fpm php8.1-cli mysql-
server-8.0
}
```

This function updates the apt package repositories and then installs all required packages.

2. Create the MySQL database that will be used by Bookstack

```
function run_database_setup() {
  mysql -h $RDS_ENDPOINT -u$DB_MASTER_USER -p$DB_MASTER_PASS <<EOL
CREATE DATABASE bookstack;
CREATE USER 'bookstack'@'%' IDENTIFIED WITH mysql_native_password BY '$DB_PASS';
GRANT ALL ON bookstack.* TO 'bookstack'@'%';FLUSH PRIVILEGES;
EOL
}
```

This function connects to the RDS database and issues the commands needed to create the database and user for bookstack as well grant the user privileges for the database and applying the changes. I used "**<<EOL**" so that I can pass multi-line text to the mysql command. This way every line passed after the mysql command will execute consecutively until "**EOL**" is typed in.

3. Download Bookstack by cloning from Git

```
function run_bookstack_download() {
  cd /var/www || exit
  git clone https://github.com/BookStackApp/BookStack.git --branch release --
single-branch bookstack
}
```

This function changes the directory to **/var/www** and clones the Bookstack repository into a single directory called "**bookstack**"

4. Install Composer (a PHP dependency manager)

```
function run_install_composer() {
  EXPECTED_CHECKSUM="$(php -r 'copy("https://composer.github.io/installer.sig",
"php://stdout");')"
  php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
  ACTUAL_CHECKSUM="$(php -r "echo hash_file('sha384', 'composer-setup.php');")"

  if [ "$EXPECTED_CHECKSUM" != "$ACTUAL_CHECKSUM" ]
  then
      >&2 echo 'ERROR: Invalid composer installer checksum'
      rm composer-setup.php
      exit 1
  fi

  php composer-setup.php --quiet
  rm composer-setup.php

  # Move composer to global installation
  mv composer.phar /usr/local/bin/composer
}
```

This function installs Composer and checks if the file is corrupted by comparing the checksums of the hashes of the downloaded file and the official checksum. If the check fails the script exits with an error message. If it succeeds, the **composer-setup.php** file executes after which it is deleted. At the end of all of this composer is moved to the **/usr/local/bin** directory so it becomes a global callable.

5. Install required PHP dependencies with Composer

```
function run_install_bookstack_composer_deps() {
  cd "$BOOKSTACK_DIR" || exit
  export COMPOSER_ALLOW_SUPERUSER=1
  php /usr/local/bin/composer install --no-dev --no-plugins
}
```

This function changes the directory to the main bookstack directory. Then it exports a global environment variable that allows composer to act with sudo privileges. After this it installs the required packages without interaction from the user.

6. Create the .env file for our Bookstack with the correct variables

```
function run_update_bookstack_env() {
  cd "$BOOKSTACK_DIR" || exit
  cp .env.example .env
  sed -i.bak "s/DB_HOST=.*$/DB_HOST=$RDS_ENDPOINT/" .env
  sed -i.bak "s@APP_URL=.*\$@APP_URL=http://$INSTANCE_IP/@" .env
```

```
  sed -i.bak 's/DB_DATABASE=.*$/DB_DATABASE=bookstack/' .env
  sed -i.bak 's/DB_USERNAME=.*$/DB_USERNAME=bookstack/' .env
  sed -i.bak "s/DB_PASSWORD=.*\$/DB_PASSWORD=$DB_PASS/" .env
  # Generate the application key
  php artisan key:generate --no-interaction --force
}
```

This function changes the directory to the Bookstack directory. Then it creates a new "**.env**" file from the example .env file. After that we use the **sed** command to substitute values in the file in-place. We achieve that using the **-i** flag and we add **.bak** at the end so that sed creates a backup file with the original values.
The sed commands works like this:
"**s/pattern_to_find/replace_with/**"
It starts with an s which stands for substitution. Then we use a slash to indicate to the command to look for a pattern to match. After that we separate with a slash again to indicate to the command what the replacement will be. Then we enclose the command with the last slash.
The whole example from above should be enclosed in quotes. It can be single quotes, but if you're using pre-defined variables in the <replace_with> part you need to use double quotes, otherwise the variable won't be read.
An example from above:

```
sed -i.bak "s/DB_HOST=.*$/DB_HOST=$RDS_ENDPOINT/" .env
```

The pattern to search is "**DB_HOST=.*$**"
The '**.**' Indicates a character/symbol/number and the '*****' after it means "one or more of the previous". The $ sign means go to the end of the line.
Then the replace part is "**DB_HOST=$RDS_ENDPOINT**" which is going to replace the previous patter with this line and read the variable from the script. The whole expression is enclosed in double quotes because we use a variable.
You might notice one of the lines is different:

```
sed -i.bak "s@APP_URL=.*\$@APP_URL=http://$INSTANCE_IP/@" .env
```

This line uses @ sings instead of forward slashes to separate the different parts. We can do that when we need to use the forward slash "**/**" inside our patter search or replace text. Again, the expression needs to be enclosed in double quotes and also the special $ sign needs to be escaped with a backwards slash "****" to has the same meaning as the previous example.

After all of this the following command generates an application key inside our **.env** file:

```
php artisan key:generate --no-interaction --force
```

Again, this command is non-interactive

7. Run Bookstack database migration

```
function run_bookstack_database_migrations() {
  cd "$BOOKSTACK_DIR" || exit
  php artisan migrate --no-interaction --force
}
```

This function goes back to our Bookstack directory. It uses the command:
"**php artisian migrate --no-interaction --force**"
This command reads our updated **.env** file and connects to the database defined inside of it after which it creates some tables and populates them with data that will be used by our Bookstack website.

8. Change file and folder permissions for Bookstack

```
function run_set_application_file_permissions() {
  cd "$BOOKSTACK_DIR" || exit
  chown -R "$SCRIPT_USER":www-data ./
  chmod -R 755 ./
  chmod -R 775 bootstrap/cache public/uploads storage
  chmod 740 .env

  git config core.fileMode false
}
```

This command goes back to the Bookstack directory. Then it changes the permissions and ownership of some files and folders. This is necessary since some of our folders will hold files for our application frontend and backend and need to be writeable by the group "**www-data**".
The last line tells git to ignore permission changes. This is to avoid annoying messages.

9. Configure the Nginx server

```
function run_configure_nginx() {
  # Set-up the required BookStack nginx config

  #Delete the default page first so we can make a new default page for bookstack
  rm -f /etc/nginx/sites-available/default
  rm -f /etc/nginx/sites-enabled/default

  #Use the tee command to read input into a new file and suppress the message
from showing in the terminal
  tee /etc/nginx/sites-available/default <<EOL >/dev/null
server {
  listen 80 default_server;
  listen [::]:80 default_server;
```

```
  server_name http://$INSTANCE_IP/;

  root /var/www/bookstack/public;
  index index.php index.html;

  location / {
    try_files \$uri \$uri/ /index.php?\$query_string;
  }

  location ~ \.php$ {
    include snippets/fastcgi-php.conf;
    fastcgi_pass unix:/run/php/php8.1-fpm.sock;
  }
}
EOL

  #Create a symbolic link for our new file
  ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/

  #Restart the nginx service for the changes to take effect
  systemctl restart nginx
}
```

This is the last function. It starts by deleting the default nginx configuration.
After that we use **tee** to write multiple lines to the command line and insert them into our new default configuration file and also pipe the output to **/dev/null** since whatever is written on the command line with the **tee** command is basically displayed twice.
The configuration block is already defined in the script to be working on the public IP of the EC2 instance.

Then we create a soft link from sites-available to sites-enabled and we restart nginx.

That is all for the application script.

Now back to the infrastructure script there were a few steps left. I will show the code and explain all of them:

```
RDS_ENDPOINT=$(aws rds describe-db-instances \
--db-instance-identifier $RDS_ID | yq '.DBInstances[0].Endpoint.Address')

sed -i "s/DB_MASTER_USER=.*$/DB_MASTER_USER=$DB_MASTER_USER/"
application_script.bash
```

```
sed -i "s/DB_MASTER_PASS=.*$/DB_MASTER_PASS=$DB_MASTER_PASS/"
application_script.bash
sed -i "s/RDS_ENDPOINT=.*$/RDS_ENDPOINT=$RDS_ENDPOINT/" application_script.bash

EC2_IP=$(aws ec2 describe-instances \
--instance-ids $EC2_ID | yq '.Reservations[0].Instances[0].PublicIpAddress')

sftp -i as2-key.pem ubuntu@$EC2_IP <<EOL
put application_script.bash
EOL

ssh -i as2-key.pem -T ubuntu@$EC2_IP << "EOL"
chmod +x application_script.bash
echo "Application script will begin executing in 5 seconds"
echo""
sleep 5
echo""
sudo ./application_script.bash $(curl -s https://checkip.amazonaws.com)
EOL

echo "The application has been successfully executed"

echo "You can check the application at http://$EC2_IP"
```

To start off we first get the Endpoint URL of our RDS database and store it in a variable using the describe command.

After this we use our predefined variables along with the endpoint and the **sed** command, much like in the application script, to update the application script itself before sending it to the EC2 instance.

After this we get the EC2 instance IP again with a describe command and store it in a variable.

After this we use SFTP to connect to the EC2 instance and send the file. We do that much like and **ssh** command after which we just use <**put filename**> to send the file.

Lastly we use the **ssh** command with **<<EOL** to connect to the EC2 instance and execute some commands. In our case we make the application script executable and then run it with sudo privileges and providing the EC2 IP using a **curl** command and the Amazon AWS api that gets the public IP of an EC2 instance when used inside one.
It is important to enclose the **EOL** in quotes for the ssh command so that the curl command in brackets will execute on the server side and not locally.

After all of this the application should be accessible on htttp://<EC2_PUBLIC_IP>/

Congratulations, you set up Bookstack and your AWS infrastructure with just scripts.

References:
https://docs.aws.amazon.com/cli/index.html
yq - yq (gitbook.io)

Full infrastructure script:

```
#!/bin/bash

### Infrastructure script for AWS CLI ###

### Define the database master username and password ###

DB_MASTER_USER="admin"
DB_MASTER_PASS="adminpass"

### Create VPC and store the vpc id in a variable ###

echo "Creating VPC"
echo""
VPC_ID=$(aws ec2 create-vpc \
--cidr-block 10.0.0.0/16 \
--tag-specification ResourceType=vpc,Tags=['{Key=Name,Value=as2-tamim-vpc}'] | yq
'.Vpc.VpcId')

### Create Subnets and store their IDs ###

echo "Creating public subnet 1"
echo""
SUBNET1=$(aws ec2 create-subnet \
--vpc-id $VPC_ID \
--cidr-block 10.0.1.0/24 \
--availability-zone us-west-2a \
--tag-specification ResourceType=subnet,Tags=['{Key=Name,Value=as2-rds-pub-ec2}']
| yq '.Subnet.SubnetId')

echo "Creating private subnet 2"
echo""
SUBNET2=$(aws ec2 create-subnet \
--vpc-id $VPC_ID \
--cidr-block 10.0.2.0/24 \
--availability-zone us-west-2a \
```

```
--tag-specification ResourceType=subnet,Tags=['{Key=Name,Value=as2-rds-pri-1}'] |
yq '.Subnet.SubnetId')

echo "Creating private subnet 3"
echo""
SUBNET3=$(aws ec2 create-subnet \
--vpc-id $VPC_ID \
--cidr-block 10.0.3.0/24 \
--availability-zone us-west-2b \
--tag-specification ResourceType=subnet,Tags=['{Key=Name,Value=as2-rds-pri-2}'] |
yq '.Subnet.SubnetId')

### Modify the first subnet to be public ###

aws ec2 modify-subnet-attribute \
--subnet-id $SUBNET1 \
--map-public-ip-on-launch

### Create Internet Gateway and store the id in a variable ###

echo "Creating Internet Gateway"
echo""
IGW_ID=$(aws ec2 create-internet-gateway \
--tag-specification ResourceType=internet-gateway,Tags=['{Key=Name,Value=as2-
igw}'] | yq '.InternetGateway.InternetGatewayId')

### Attach the Internet Gateway to the VPC ###

echo "Attaching Internet Gateway to the VPC"
echo""
aws ec2 attach-internet-gateway \
--internet-gateway-id $IGW_ID \
--vpc-id $VPC_ID >/dev/null

### Create a route table and store the id in a variable ###

echo "Creating route table"
echo""
ROUTE_TABLE_ID=$(aws ec2 create-route-table \
--vpc-id $VPC_ID \
--tag-specification ResourceType=route-table,Tags=['{Key=Name,Value=as2-rt}'] |
yq '.RouteTable.RouteTableId')

### Create a route to the Internet Gateway ###
```

```
echo "Creating IGW route for the route table"
echo""
aws ec2 create-route \
--route-table-id $ROUTE_TABLE_ID \
--destination-cidr-block 0.0.0.0/0 \
--gateway-id $IGW_ID >/dev/null

### Associate the route table with the first subnet ###

echo "Associating the public subnet with the route table"
echo""
aws ec2 associate-route-table \
--subnet-id $SUBNET1 \
--route-table-id $ROUTE_TABLE_ID >/dev/null

### Create a security group and store the id in a variable ###

echo "Creating a security group for the EC2 instance"
echo""
SECURITY_GROUP_ID=$(aws ec2 create-security-group \
--group-name rds-ec2-sg \
--description "Security group for EC2 in Assignment 2. Allows SSH and HTTP from
anywhere." \
--vpc-id $VPC_ID | yq '.GroupId')

### Add a rule to the security group to allow SSH from anywhere ###

echo "Adding inbound rule for the EC2 SG for SSH access from anywhere"
echo""
aws ec2 authorize-security-group-ingress \
--group-id $SECURITY_GROUP_ID \
--protocol tcp --port 22 \
--cidr 0.0.0.0/0 >/dev/null

### Add a rule to the security group to allow HTTP from anywhere ###

echo "Adding inbound rule for the EC2 SG for HTTP access from anywhere"
echo""
aws ec2 authorize-security-group-ingress \
--group-id $SECURITY_GROUP_ID \
--protocol tcp --port 80 \
--cidr 0.0.0.0/0 >/dev/null

### Create a key pair and store the private key in a file ###
```

```
echo "Creating key pair for the EC2 instance"
echo""
aws ec2 create-key-pair \
--key-name as2-ec2-key \
--key-type ed25519 \
--query 'KeyMaterial' \
--output text > as2-key.pem

### Change the permissions of the private key file ###

echo "Changing the permissions of the private key to 400"
echo""
chmod 400 as2-key.pem

### Create an EC2 Ubuntu instance using the created key pair and security group
and store its instance ID ###

echo "Creating an EC2 Ubuntu instance"
echo""
EC2_ID=$(aws ec2 run-instances \
--image-id ami-0735c191cf914754d \
--count 1 \
--instance-type t2.micro \
--key-name as2-ec2-key \
--security-group-ids $SECURITY_GROUP_ID \
--subnet-id $SUBNET1 \
--associate-public-ip-address \
--tag-specification ResourceType=instance,Tags=['{Key=Name,Value=as2-rds-ec2}'] |
yq '.Instances[0].InstanceId')

### Wait for the instance to be running ###

aws ec2 wait instance-running \
--instance-ids $EC2_ID

echo "EC2 instance is up and running"
echo""
### Create an RDS subnet group and store the subnet group name in a variable ###

echo "Creating RDS Subnet Group"
echo""
RDS_SUBNET_GROUP_NAME=$(aws rds create-db-subnet-group \
--db-subnet-group-name as2-rds-subnet-group \
--db-subnet-group-description "Subnet group for RDS in Assignment 2." \
--subnet-ids $SUBNET2 $SUBNET3 | yq '.DBSubnetGroup.DBSubnetGroupName')
```

```
### Create a security group for the database and store the id in a variable ###

echo "Creating a security group for the RDS database"
echo""
RDS_SECURITY_GROUP_ID=$(aws ec2 create-security-group \
--group-name rds-sg \
--description "Security group for RDS in Assignment 2. Allows MySQL access from
the VPC only." \
--vpc-id $VPC_ID | yq '.GroupId')

### Add a rule to the security group to allow MySQL access from the VPC CIDR only
###

echo "Adding an inbound rule to the RDS DB SG for MySQL access from the VPC"
echo""
aws ec2 authorize-security-group-ingress \
--group-id $RDS_SECURITY_GROUP_ID \
--protocol tcp --port 3306 \
--cidr 10.0.0.0/16 >/dev/null

### Create an RDS MySQL instance using the created subnet group and security
group. Store the instance identifier ###

echo "Creating an RDS MySQL database"
echo""
echo "This step takes the longest time to complete"
echo "You will be notified when the database is up and running"
echo""
RDS_ID=$(aws rds create-db-instance \
--db-name as2_rds \
--db-instance-identifier as2-rds \
--db-instance-class db.t3.micro \
--engine mysql \
--master-username "$DB_MASTER_USER" \
--master-user-password "$DB_MASTER_PASS" \
--allocated-storage 20 \
--vpc-security-group-ids $RDS_SECURITY_GROUP_ID \
--db-subnet-group-name $RDS_SUBNET_GROUP_NAME \
--no-publicly-accessible \
--engine-version 8.0.28 \
--storage-type gp2 \
--availability-zone us-west-2a | yq '.DBInstance.DBInstanceIdentifier')

### Wait for the instance to be available ###
```

```
aws rds wait db-instance-available \
--db-instance-identifier $RDS_ID

echo "RDS database is up and running"
echo""

### Get the RDS endpoint and store it in a variable ###

echo "Getting the RDS endpoint and storing it in a variable"

RDS_ENDPOINT=$(aws rds describe-db-instances \
--db-instance-identifier $RDS_ID | yq '.DBInstances[0].Endpoint.Address')

echo""
echo "The infrastructure has been successfully created"

### Update the database username, password and endpoint variables in the
application script ###

echo""
echo "Updating the database username, password and endpoint variables in the
application script"

sed -i "s/DB_MASTER_USER=.*$/DB_MASTER_USER=$DB_MASTER_USER/"
application_script.bash
sed -i "s/DB_MASTER_PASS=.*$/DB_MASTER_PASS=$DB_MASTER_PASS/"
application_script.bash
sed -i "s/RDS_ENDPOINT=.*$/RDS_ENDPOINT=$RDS_ENDPOINT/" application_script.bash

echo""
echo "The application script has been updated"

### Copy the application script to the EC2 instance using SFTP ###
### Get the public IP of the EC2 instance and store it in a variable ###

echo""
echo "Getting the public IP of the EC2 instance"

EC2_IP=$(aws ec2 describe-instances \
--instance-ids $EC2_ID | yq '.Reservations[0].Instances[0].PublicIpAddress')

echo""
echo "Copying the application script to the EC2 instance"
```

```
echo "You will be prompted to type in 'yes' to continue connecting to the EC2 instance"
echo "This is only needed the first time you connect to the EC2 instance. It is a security measure"
echo "Type in 'yes' and press enter to continue"
echo""
echo""

sftp -i as2-key.pem ubuntu@$EC2_IP <<EOL
put application_script.bash
EOL
echo""

### Run the application script on the EC2 instance ###

echo "Running the application script on the EC2 instance"
echo""
echo""

ssh -i as2-key.pem -T ubuntu@$EC2_IP << "EOL"
chmod +x application_script.bash
echo "Application script will begin executing in 5 seconds"
echo""
sleep 5
echo""
sudo ./application_script.bash $(curl -s https://checkip.amazonaws.com)
EOL

echo""
echo "The application has been successfully executed"

echo""
echo "You can check the application at http://$EC2_IP"

echo""
echo "Congratulations! You have successfully completed Assignment 2!"
### Completed ###
```

Full application script:

```
#!/bin/bash

echo "This script installs a new BookStack instance on a fresh Ubuntu 22.04 server."
echo "This script does not ensure system security."
```

```
echo ""

# Generate a path for a log file to output into for debugging
LOGPATH=$(realpath "bookstack_install_$(date +%s).log")

# Get the current user running the script
SCRIPT_USER="${SUDO_USER:-$USER}"

# Generate a password for the database
DB_PASS="$(head /dev/urandom | tr -dc A-Za-z0-9 | head -c 13)"

# The directory to install BookStack into
BOOKSTACK_DIR="/var/www/bookstack"

#Get the EC2 Instance public IP
INSTANCE_IP=$(curl -s https://checkip.amazonaws.com)

# Get the domain from the arguments (Requested later if not set)
DOMAIN=$1

# Set the RDS master username, password and endpoint
# These will be automaticall substituted to the correct values in the
infrastructure script
DB_MASTER_USER=EnterUsernameHere
DB_MASTER_PASS=EnterPasswordHere
RDS_ENDPOINT=EnterEndpointHere

# Prevent interactive prompts in applications
export DEBIAN_FRONTEND=noninteractive

# Echo out an error message to the command line and exit the program
# Also logs the message to the log file
function error_out() {
  echo "ERROR: $1" | tee -a "$LOGPATH" 1>&2
  exit 1
}

# Echo out an information message to both the command line and log file
function info_msg() {
  echo "$1" | tee -a "$LOGPATH"
}

# Run some checks before installation to help prevent messing up an existing
# web-server setup.
function run_pre_install_checks() {
```

```
  # Check we're running as root and exit if not
  if [[ $EUID -gt 0 ]]
  then
    error_out "This script must be ran with root/sudo privileges"
  fi

  # Check if Nginx appears to be installed and exit if so
  if [ -d "/etc/nginx/sites-enabled" ]
  then
    error_out "This script is intended for a fresh server install, existing nginx
config found, aborting install"
  fi

  # Check if MySQL appears to be installed and exit if so
  if [ -d "/var/lib/mysql" ]
  then
    error_out "This script is intended for a fresh server install, existing MySQL
data found, aborting install"
  fi
}

# Fetch domain to use from first provided parameter,
# Otherwise request the user to input their domain
function run_prompt_for_domain_if_required() {
  if [ -z "$DOMAIN" ]
  then
    info_msg ""
    info_msg "Enter the domain (or IP if not using a domain) you want to host
BookStack on and press [ENTER]."
    info_msg "Examples: my-site.com or docs.my-site.com or $(curl -s
https://checkip.amazonaws.com)"
    read -r DOMAIN
  fi

  # Error out if no domain was provided
  if [ -z "$DOMAIN" ]
  then
    error_out "A domain must be provided to run this script"
  fi
}

# Install core system packages
function run_package_installs() {
  apt update
```

```
  apt install -y git unzip nginx php8.1 curl php8.1-curl php8.1-mbstring php8.1-
ldap \
  php8.1-xml php8.1-zip php8.1-gd php8.1-mysql php8.1-fpm php8.1-cli mysql-
server-8.0
}

# Set up database
function run_database_setup() {
  mysql -h $RDS_ENDPOINT -u$DB_MASTER_USER -p$DB_MASTER_PASS <<EOL
CREATE DATABASE bookstack;
CREATE USER 'bookstack'@'%' IDENTIFIED WITH mysql_native_password BY '$DB_PASS';
GRANT ALL ON bookstack.* TO 'bookstack'@'%';FLUSH PRIVILEGES;
EOL
}

# Download BookStack
function run_bookstack_download() {
  cd /var/www || exit
  git clone https://github.com/BookStackApp/BookStack.git --branch release --
single-branch bookstack
}

# Install composer
function run_install_composer() {
  EXPECTED_CHECKSUM="$(php -r 'copy("https://composer.github.io/installer.sig",
"php://stdout");')"
  php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
  ACTUAL_CHECKSUM="$(php -r "echo hash_file('sha384', 'composer-setup.php');")"

  if [ "$EXPECTED_CHECKSUM" != "$ACTUAL_CHECKSUM" ]
  then
      >&2 echo 'ERROR: Invalid composer installer checksum'
      rm composer-setup.php
      exit 1
  fi

  php composer-setup.php --quiet
  rm composer-setup.php

  # Move composer to global installation
  mv composer.phar /usr/local/bin/composer
}

# Install BookStack composer dependencies
function run_install_bookstack_composer_deps() {
```

```
  cd "$BOOKSTACK_DIR" || exit
  export COMPOSER_ALLOW_SUPERUSER=1
  php /usr/local/bin/composer install --no-dev --no-plugins
}

# Copy and update BookStack environment variables
function run_update_bookstack_env() {
  cd "$BOOKSTACK_DIR" || exit
  cp .env.example .env
  sed -i.bak "s/DB_HOST=.*$/DB_HOST=$RDS_ENDPOINT/" .env
  sed -i.bak "s@APP_URL=.*\$@APP_URL=http://$INSTANCE_IP/@" .env
  sed -i.bak 's/DB_DATABASE=.*$/DB_DATABASE=bookstack/' .env
  sed -i.bak 's/DB_USERNAME=.*$/DB_USERNAME=bookstack/' .env
  sed -i.bak "s/DB_PASSWORD=.*\$/DB_PASSWORD=$DB_PASS/" .env
  # Generate the application key
  php artisan key:generate --no-interaction --force
}

# Run the BookStack database migrations for the first time
function run_bookstack_database_migrations() {
  cd "$BOOKSTACK_DIR" || exit
  php artisan migrate --no-interaction --force
}

# Set file and folder permissions
# Sets current user as owner user and www-data as owner group then
# provides group write access only to required directories.
# Hides the `.env` file so it's not visible to other users on the system.
function run_set_application_file_permissions() {
  cd "$BOOKSTACK_DIR" || exit
  chown -R "$SCRIPT_USER":www-data ./
  chmod -R 755 ./
  chmod -R 775 bootstrap/cache public/uploads storage
  chmod 740 .env

  # Tell git to ignore permission changes
  git config core.fileMode false
}

# Setup nginx with the needed modules and config
function run_configure_nginx() {
  # Set-up the required BookStack nginx config

  #Delete the default page first so we can make a new default page for bookstack
  rm -f /etc/nginx/sites-available/default
```

```
  rm -f /etc/nginx/sites-enabled/default

  #Use the tee command to read input into a new file and suppress the message
from showing in the terminal
  tee /etc/nginx/sites-available/default <<EOL >/dev/null
server {
  listen 80 default_server;
  listen [::]:80 default_server;

  server_name http://$INSTANCE_IP/;

  root /var/www/bookstack/public;
  index index.php index.html;

  location / {
    try_files \$uri \$uri/ /index.php?\$query_string;
  }

  location ~ \.php$ {
    include snippets/fastcgi-php.conf;
    fastcgi_pass unix:/run/php/php8.1-fpm.sock;
  }
}
EOL

  #Create a symbolic link for our new file
  ln -s /etc/nginx/sites-available/default /etc/nginx/sites-enabled/

  #Restart the nginx service for the changes to take effect
  systemctl restart nginx
}

info_msg "This script logs full output to $LOGPATH which may help upon issues."
sleep 1

run_pre_install_checks
run_prompt_for_domain_if_required
info_msg ""
info_msg "Installing using the domain or IP \"$DOMAIN\""
info_msg ""
sleep 1

info_msg "[1/9] Installing required system packages... (This may take several
minutes)"
run_package_installs >> "$LOGPATH" 2>&1
```

```
info_msg "[2/9] Preparing MySQL database..."
run_database_setup >> "$LOGPATH" 2>&1

info_msg "[3/9] Downloading BookStack to ${BOOKSTACK_DIR}..."
run_bookstack_download >> "$LOGPATH" 2>&1

info_msg "[4/9] Installing Composer (PHP dependency manager)..."
run_install_composer >> "$LOGPATH" 2>&1

info_msg "[5/9] Installing PHP dependencies using composer..."
run_install_bookstack_composer_deps >> "$LOGPATH" 2>&1

info_msg "[6/9] Creating and populating BookStack .env file..."
run_update_bookstack_env >> "$LOGPATH" 2>&1

info_msg "[7/9] Running initial BookStack database migrations..."
run_bookstack_database_migrations >> "$LOGPATH" 2>&1

info_msg "[8/9] Setting BookStack file & folder permissions..."
run_set_application_file_permissions >> "$LOGPATH" 2>&1

info_msg "[9/9] Configuring nginx server..."
run_configure_nginx >> "$LOGPATH" 2>&1

info_msg "------------------------------------------------------------"
info_msg "Setup finished, your BookStack instance should now be installed!"
info_msg "- Default login email: admin@admin.com"
info_msg "- Default login password: password"
info_msg "- Access URL: http://localhost/ or http://$DOMAIN/"
info_msg "- BookStack install path: $BOOKSTACK_DIR"
info_msg "- Install script log: $LOGPATH"
info_msg "------------------------------------------------------------"
```