Application Portability Immersion Day

Lab Guide

[Introduction 1](#_Toc523149648)

[Prerequisites 2](#_Toc523149649)

[Module 1: Install REST API components using AWS Lambda, Amazon RDS, and Amazon MQ 3](#_Toc523149650)

[Module 2: Install REST API components using kubeless, Amazon RDS, and Amazon MQ 6](#_Toc523149651)

[Module 3: Migrating Database to a VM 9](#_Toc523149652)

[Module 4: Replace Messaging Component with Amazon SQS 11](#_Toc523149653)

# Introduction

This lab explores porting components of a REST API service between managed AWS services like Lambda, RDS, and SQS to non-managed equivalents. After you complete this lab, you should understand the tradeoffs involved for components that use well-defined standards, emerging standards, or no standards.

Figure 1 shows the simple system we’ll use for this lab. The system includes an API endpoint, the implementation of the API, a database for storage, and a messaging queue used to notify other services about item creation.

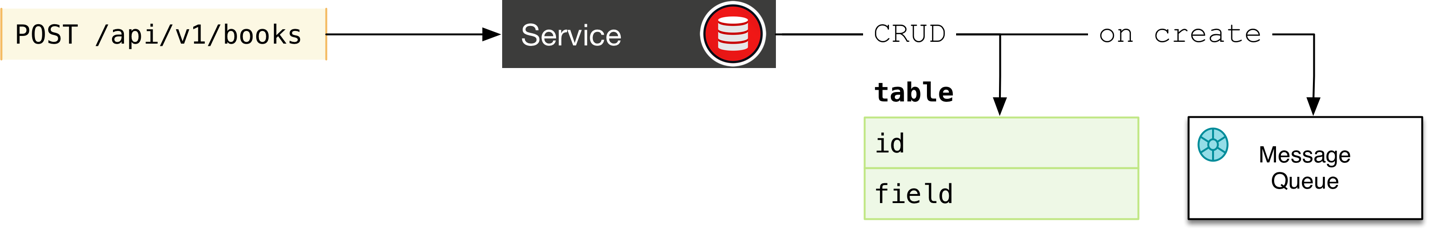


Figure 1: REST API service including API endpoint, service implementation, database, and messaging service.

# Prerequisites

Let’s set up the basic lab infrastructure.

This lab works with Linux or Mac machines. If you have a Windows machine, please [create an Amazon Linux EC2 instance](https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html) and run the lab from that machine.

| Step | Instructions |
| --- | --- |
| Install and configure the AWS CLI | Follow the standard instructions to [install](https://docs.aws.amazon.com/cli/latest/userguide/installing.html) and [configure](https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html) the AWS CLI. When you run the aws configure command, be sure to set your region. This lab should work in us-west-2, us-east-1, and other regions where EKS is available. If you are running the lab on an EC2 instance and wish to use an IAM role rather than your IAM access keys, you will need a role that has full permission to the following services: Amazon MQ, EKS, EC2, S3, RDS, IAM, VPC, CloudFormation, Lambda, and API Gateway. You also need permission to create a CloudWatch log group. |
| Install npm | Follow the instructions for your platform at <https://www.npmjs.com/get-npm>. For Amazon Linux and other common Linux variants you can use: <https://nodejs.org/en/download/package-manager/#enterprise-linux-and-fedora>. |
| Install eksctl | On Mac: brew install weaveworks/tap/eksctl  On Linux: Follow instructions from <https://eksctl.io/>:  curl --silent --location "https://github.com/weaveworks/eksctl/releases/download/latest\_release/eksctl\_$(uname -s)\_amd64.tar.gz" | tar xz -C /tmp  sudo mv /tmp/eksctl /usr/local/bin |
| Download the lab material from the lab bucket | export WORKDIR=<set a working path to use for the lab>  mkdir $WORKDIR  wget -O labs.zip “<https://portable-app-lab.s3.amazonaws.com/labs.zip?AWSAccessKeyId=AKIAJAV7J4ZSTOAPQPGQ&Expires=1541444560&Signature=Q8Mq6tYiqE9csPusyCR5z%2FFXwhI%3D>”[[1]](#footnote-1)  unzip labs.zip  mv portable-app-lab/\* $WORKDIR/ |
| Install the serverless framework | sudo npm install -g serverless |
| Download and configure the EKS authenticator | * mkdir $HOME/bin * Install aws-iam-authenticator   + Mac: curl -o $HOME/bin/aws-iam-authenticator <https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/darwin/amd64/aws-iam-authenticator>   + Linux: curl -o $HOME/bin/aws-iam-authenticator <https://amazon-eks.s3-us-west-2.amazonaws.com/1.10.3/2018-07-26/bin/linux/amd64/aws-iam-authenticator> * chmod +x $HOME/bin/aws-iam-authenticator * echo 'export PATH=$HOME/bin:$PATH' >> ~/.bash\_profile |
| Install kubectl | * Follow instructions at <https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html#install-kubectl-linux> |
| Create your EKS cluster | * eksctl create cluster –n workshop * Edit ~/.kube/config and confirm that the authenticator command is set to aws-iam-authenticator |
| Install kubeless | * export RELEASE=$(curl -s https://api.github.com/repos/kubeless/kubeless/releases/latest | grep tag\_name | cut -d '"' -f 4) * kubectl create ns kubeless * kubectl create -f <https://github.com/kubeless/kubeless/releases/download/$RELEASE/kubeless-$RELEASE.yaml> * export OS=$(uname -s| tr '[:upper:]' '[:lower:]') * curl -OL https://github.com/kubeless/kubeless/releases/download/$RELEASE/kubeless\_$OS-amd64.zip && unzip kubeless\_$OS-amd64.zip && sudo mv bundles/kubeless\_$OS-amd64/kubeless /usr/local/bin/   See <https://kubeless.io/docs/quick-start/> for more information. |
| Install ingress controller | * kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/mandatory.yaml> * kubectl apply -f <https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/provider/aws/service-l4.yaml> * kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/provider/aws/patch-configmap-l4.yaml   In this case we use Nginx as a basic ingress controller for request routing. However, we could also use more full-featured options like [Kong](https://github.com/Kong/kubernetes-ingress-controller/blob/master/deploy/README.md), which provides additional API gateway features. |
| Install jq | * sudo yum install -y jq |

# Module 1: Install REST API components using AWS Lambda, Amazon RDS, and Amazon MQ

In this section you’ll deploy the API service handler, a database, and a message broker as shown in Figure 2.

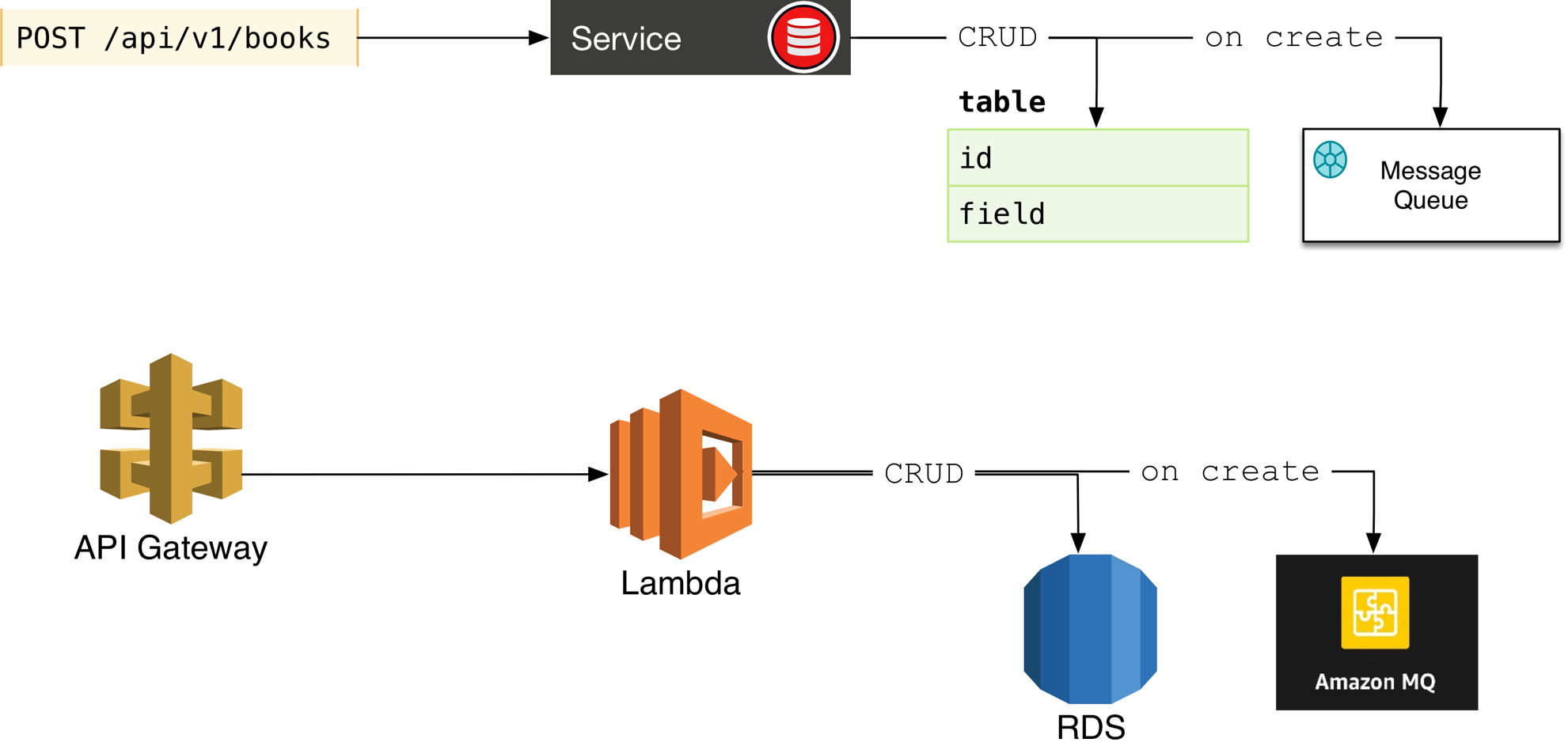


Figure 2: Baseline implementation using endpoints on APIO Gateway, service logic in Lambda, an RDS database, and Apache MQ for messaging.

| Step | Instructions |
| --- | --- |
| Review CloudFormation parameters | Open $WORKDIR/scripts/create.sh in a text editor and review all the parameters. You will need to modify the following at a minimum:   * AllowedCidrIngress (Line 65) – Choose the CIDR of the network that you connect to AWS from. * Keyname (Line 78) – Choose your own SSH key in your AWS account   Copy any changed parameter values to $WORKDIR/scripts/update.sh. |
| Deploy CloudFormation stack | cd $WORKDIR  export MYBUCKET=<pick a unique bucket name>  export MYSTACK=<pick a unique stack name>  export REGION=us-west-2 # or pick another region  aws s3 mb s3://$MYBUCKET  chmod +x ./scripts/create.sh  ./scripts/create.sh $MYBUCKET cfn $MYSTACK $REGION  Wait for the stack to complete before continuing. |
| Start SSH proxy for Apache MQ | Follow steps 4 and 5 in the [instructions](https://docs.aws.amazon.com/amazon-mq/latest/developer-guide/accessing-web-console-of-broker-without-private-accessibility.html) to configure a proxy in your browser on your local workstation.  Go to the CloudFormation console and find the nested stack ‘<stack prefix>-JumpStack-\*\*\*\*’. In the Outputs tab, note the IP address of the jump host. Then run:  ssh -i <your ssh private key> -N -C -q -f -D 8080 ec2-user@<jump host IP>  Now open the MQ web console link from the Amazon MQ console. |
| Set environment variables | Go to the Amazon MQ console and select the broker you created in the last step. Note the ActiveMQ web console URLs and use them to set the following environment variables:  export MQ\_1=<first URL without protocol or port number>  export MQ\_2=<second URl without protocol or port number>  export MQ\_Q=userqueue  Omit the protocol ‘https://’ and the port (8162). |
| Go into the working directory | mkdir $WORKDIR/local  cd $WORKDIR/local |
| Install the serverless template | serverless create --template aws-python --path restapi-aws |
| Go to new template directory | cd restapi-aws |
| Install packaging plugin | npm install --save serverless-python-requirements |
| Copy files from lab bucket | cp $WORKDIR/restapi-aws/handler.py .  cp $WORKDIR/restapi-aws/new\_user.json .  cp $WORKDIR/restapi-aws/requirements.txt .  cp $WORKDIR/restapi-aws/serverless.yml . |
| Edit serverless.yml for your CFN stack | In serverless.yml, replace ‘PortableStack’ with the name of your CFN stack created in the prerequisites. |
| Deploy the API | serverless deploy -v |
| Test function to create schema | serverless invoke -f reset -l  If this function succeeds, it creates the database schema. You can verify the schema by logging in to the jump host, installing the mysql client, connecting to the database, and showing all the tables. The database credentials are parameters in the CloudFormation templates; the default user name is ‘dbuser’, the default password is ‘Mydbcred01’, and the database name is ‘portabledb’. |
| Test function to create user | serverless invoke -f create -p new\_user.json -l  The JSON file specifies a user ID and an email address. If you run this command again, be sure to update the JSON file to use a unique user ID. |
| Test function to list all users | serverless invoke -f get -l  Confirm that your user was created by querying the ‘users’ table in the database. |
| Verify messages going into MQ broker | In the MQ web console, open the queue graph. You should see messages queued in the userqueue queue. The user name and password for Apache MQ are set in the Cloud Formation templates; the default values are ‘mquser’ and ‘Mymqbrokercred01’. You can also find these by going to the Amazon MQ console, selecting your broker, and reviewing the ‘Users’ section. |

# Module 2: Install REST API components using kubeless, Amazon RDS, and Amazon MQ

As the first step towards transitioning our REST API, let’s move the API endpoints and service handlers to kubernetes. We’ll continue to use the serverless framework to deploy our service handlers, but target our EKS kubernetes cluster instead of Lambda (Figure 3). This step is a good example of using a framework like serverless.com to manage portability in an area where standards are nascent or emerging.

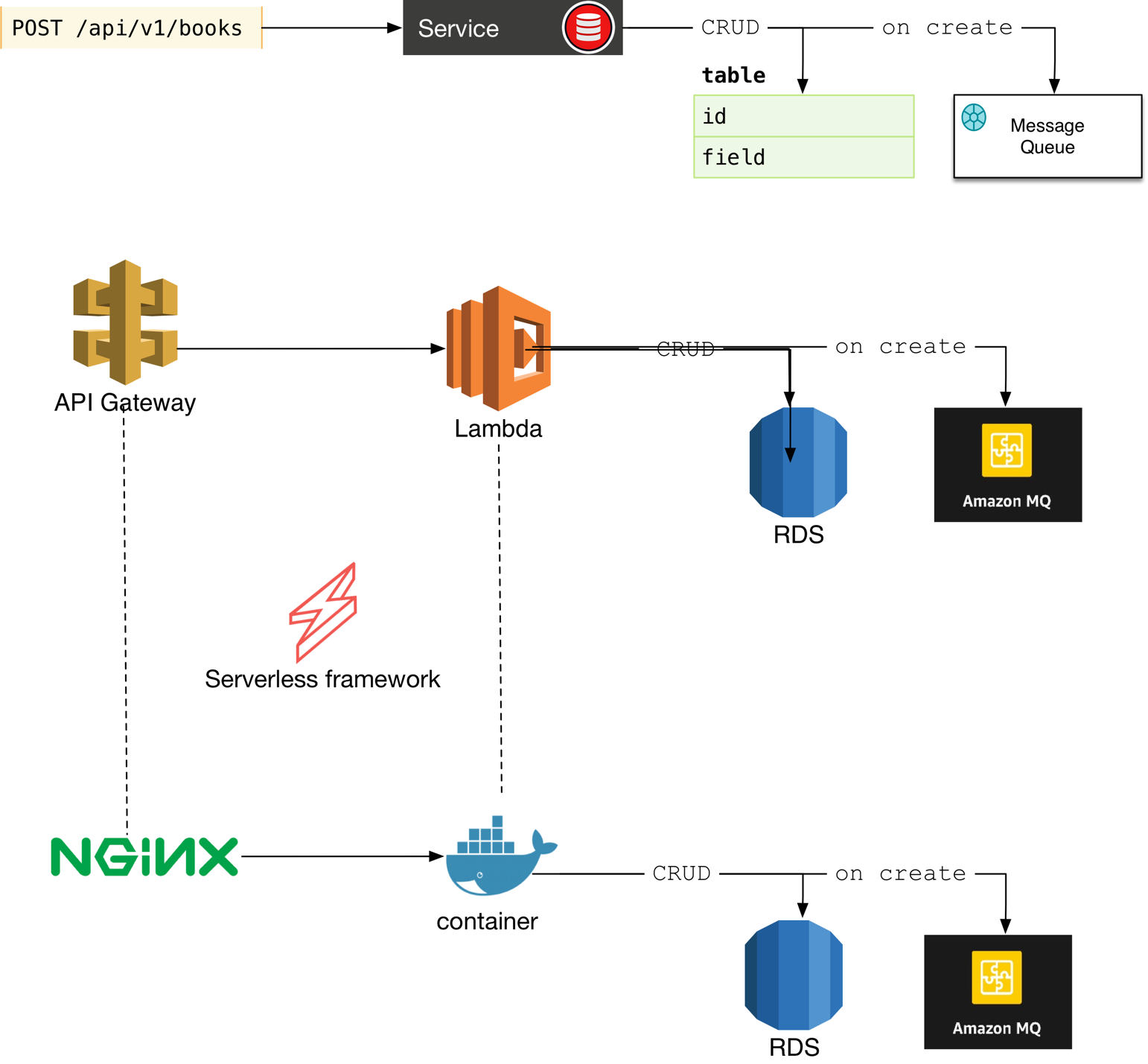


Figure 3: Replacing Lambda with functions running on kubernetes.

| Step | Instructions |
| --- | --- |
| Go into the working directory | cd $WORKDIR/local |
| Install the serverless template | serverless create --template kubeless-python --path restapi-kube |
| Go to new template directory | cd restapi-kube |
| Install the dependencies | npm install |
| Install the kubeless plugin | npm install –save serverless-kubeless |
| Copy files from lab bucket | cp $WORKDIR/restapi-kube/handler.py .  cp $WORKDIR/restapi-kube/new\_user.json .  cp $WORKDIR/restapi-kube/requirements.txt .  cp $WORKDIR/restapi-kube/serverless.yml . |
| Patch the kubeless plugin to support EKS | cp $WORKDIR/restapi-kube/helpers.js node\_modules/serverless-kubeless/lib/helpers.js |
| Set environment variables | From the file $WORKDIR/scripts/create.sh, identify the RDS database name, database user name, and database password, and use them to set the environment variables DBName, DBUser, and DBPass. Find the Amazon MQ broker user name and broker password and use them to set the environment variables BrokerUser and BrokerPass. Finally, from the output of the master CloudFormation stack, find the output DBClusterEndpoint and use that value to set an environment variable of the same name. |
| Deploy the API | serverless deploy -v |
| Peer the VPCs to allow the functions running in kubernetes to communicate to the RDS database | * Identify the VPC ID of the VPC tagged with Project=PortableApps and the VPC tagged with Name=EKS-workshop-VPC-VPC. * Create a peering request and note the peering request ID in the output: aws ec2 create-vpc-peering-connection --peer-vpc-id <ID of VPC used for RDS> --vpc-id <ID of VPC used for EKS> * Accept the peering request: aws ec2 accept-vpc-peering-connection --vpc-peering-connection-id <peering request ID> * Modify the route tables for both VPCs and add a route to the other VPC’s CIDR block using the peering connection as the target. |
| Modify security groups to let k8s nodes communicate with database and MQ broker | * Go to the EC2 console and select Security Groups. * Add a rule to the BrokerFirewall group allowing inbound connections to port 61614 from the security group whose name starts with EKS-workshop-DefaultNodeGroup-NodeSecurityGroup. * Add a rule to the DBFirewall group allowing inbound connections to port 3306 from the security group whose name starts with EKS-workshop-DefaultNodeGroup-NodeSecurityGroup. |
| Test get method | export MY\_INGRESS=$(kubectl get svc ingress-nginx -n ingress-nginx -o json | jq -r .status.loadBalancer.ingress[0].hostname)  curl -d '{"kubeless": "on AWS"}' -H "Host: myk8s.io" -H "Content-Type:application/json" ${MY\_INGRESS}/users/get |
| Test create method | curl -d @new\_user.json -H "Host: myk8s.io" -H "Content-Type:application/json" ${MY\_INGRESS}/users/create |
| Verify messages going into MQ broker | In the MQ web console, open the queue graph. You should see messages queued in the userqueue queue. |

What changes did you notice in the serverless.yml descriptor and the source code?

# Module 3: Migrating Database to a VM

In this module we’ll demonstrate moving the database from Amazon RDS to an EC2 instance (VM), as shown in Figure 4. We could also move the database on-premises to target a different relational database engine like PostgreSQL. This step is a good example of easily swapping back-end engines when standards like SQL for relational databases are well defined.

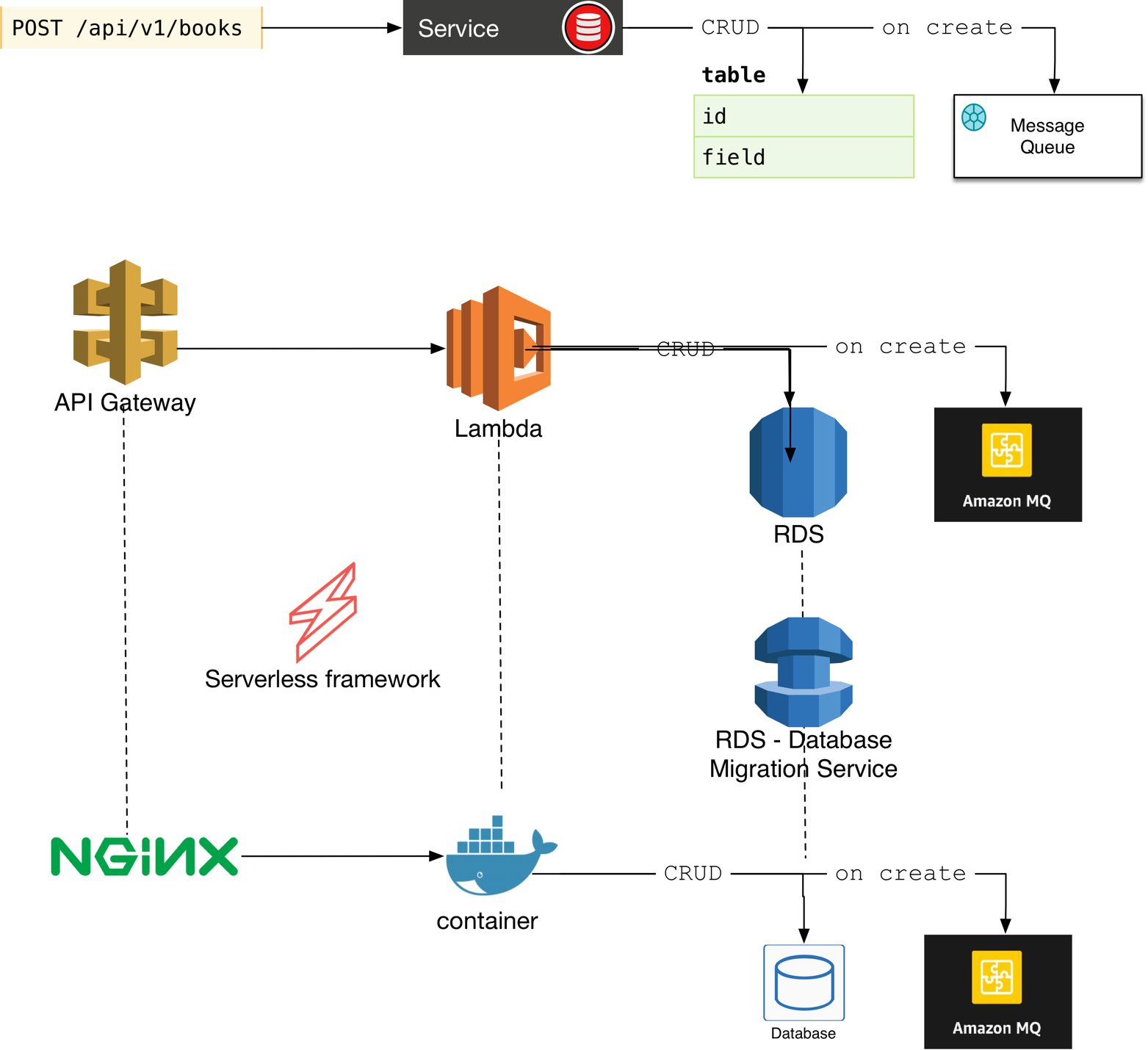


Figure 4: Replacing RDS with standalone database

| Step | Instructions |
| --- | --- |
| Launch a MySQL EC2 instance. | In the EC2 console, launch an instance using these settings.   * AMI: search for **bitnami-mysql-5.6.40-0-linux-debian-9-x86\_64-hvm-ebs.** * Instance type: t2.large * Use the VPC deployed in the CloudFormation stack, and choose SubnetPrivateA. * Allocate a 50 GB EBS volume. * Choose the DBFirewall security group. * Use the keypair you used for the CloudFormation template. |
| Find the MySQL credentials. | SSH to the jump host using SSH agent forwarding, and then SSH to the MySQL instance using its private IP address. In order to connect over SSH, add a rule to the DBFirewall group allowing SSH access from the ClientFirewall security group. The database credentials are listed in /home/bitnami/bitnami\_credentials.   * (on your workstation) ssh-add <SSH key> * (on your workstation) ssh -I <ssh private key> -A ec2-user@<jump host IP> * (on jump host) ssh bitnami@<MySQL IP> |
| Set up a Database Migration Service (DMS) subnet group. | Go to the DMS console. Create a subnet group and choose the two private subnets from the VPC created in the CloudFormation stack. |
| Create a DMS replication instance. | In the DMS console, create a replication instance. Use the VPC from the CloudFormation stack and the subnet group you just created. Choose the ClientFirewall. Leave other settings as default. |
| Create a source DMS endpoint. | In the DMS console, create an endpoint of type Source. Choose the RDS cluster and use any node from that cluster. Provide the database credentials. |
| Create a target DMS endpoint. | In the DMS console, create an endpoint of type Target. Choose MySQL and provide the private IP address of the MySQL EC2 instance. Provide the database credentials. |
| Create a migration task. | In the DMS console, create a full replication task. Select the schema ‘portabledb’. Wait for this task to complete. |
| Verify that data was migrated. | In the SSH connection to the MySQL server, connect to the local database and run a SQL query to show all entries from the users table. The results should match the data in the RDS database. |
| (Optional) Update handler to use new database | In order to use the new database, reset the DBUser, DBPass, and DBClusterEndpoint environment variables to point to the new MySQL instance. Redeploy the restapi-kube function using ‘serverless deploy -v’, and test the function handlers again. |

Note that we did not have to change any code or even the serverless.yml definition. We only had to point at a new database URL.

# Module 4: Replace Messaging Component with Amazon SQS

Finally, let’s use Amazon SQS rather than Amazon MQ as the messaging broker (Figure 5). In this case there are no standards for messaging services in general, although some protocols like JMS do have defined standards.

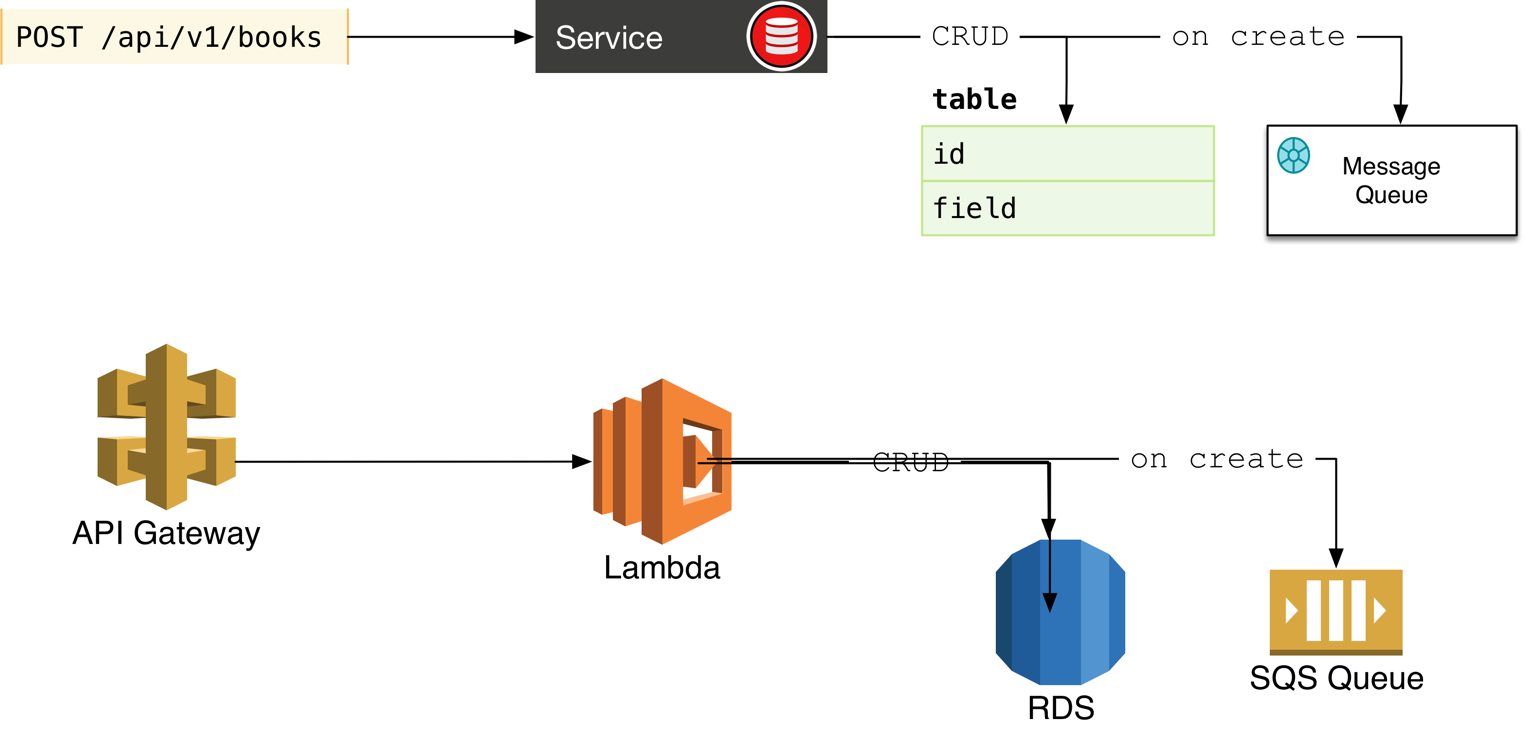


Figure 5: Replacing the messaging service.

| Step | Instructions |
| --- | --- |
| Go into the working directory | cd $WORKDIR/local |
| Install the serverless template | serverless create --template aws-python --path restapi-aws-sqs |
| Go to new template directory | cd restapi-aws-sqs |
| Install packaging plugin | npm install --save serverless-python-requirements |
| Copy files from lab bucket | cp $WORKDIR/restapi-aws-sqs/handler.py .  cp $WORKDIR/restapi-aws-sqs/new\_user.json .  cp $WORKDIR/restapi-aws-sqs/requirements.txt .  cp $WORKDIR/restapi-aws-sqs/serverless.yml . |
| Create an SQS queue. | Go to the SQS console and create a standard queue. Note the queue name and ARN. |
| Set SQS environment variables | export SQS\_ARN=<SQS queue ARN>  export SQS\_QUEUE=<SQS queue name> |
| Deploy the API | serverless deploy -v |
| Test function to create user | serverless invoke -f create -p new\_user.json -l  The JSON file specifies a user ID and an email address. If you run this command again, be sure to update the JSON file to use a unique user ID. |
| Verify messages going into SQS queue | In the SQS web console, preview the messages going into the queue and verify that you see one for the user you just created. |

Compare the source code for the handler in restapi-aws and restapi-aws-sqs. What differences do you see?

1. This URL is time limited and will expire on or around November 3rd, 2018. [↑](#footnote-ref-1)