CloudFormation: Build a WordPress Site

In the AWS console click **Services.**

Then click **CloudFormation** which is located under Management & Governance

Click **Create Stack**

Prepare Template: ✓Create Template in Designer ← Select Create Template in Designer

Click the **Create Template in Designer** button

★At the bottom of the page select the **Template** Tab

Choose template language: ✓YAML ← select YAML

In the new.template line 1 will say the following:

AWSTemplateFormatVersion: 2010-09-09

★For simplicity remove that line so that the file is now **completely** blank.

★Now copy and paste in the following YAML code:

```
AWSTemplateFormatVersion: 2010-09-09
Description: >-
   AWS CloudFormation Sample Template WordPress_Single_Instance: WordPress is
web
   software you can use to create a beautiful website or blog. This template
   installs WordPress with a local MySQL database for storage. It
demonstrates
   using the AWS CloudFormation bootstrap scripts to deploy WordPress.
   **WARNING** This template creates an Amazon EC2 instance. You will be
billed
   for the AWS resources used if you create a stack from this template.
   Modified Feb 2020 to build unrestricted on Linux2, plus add automation to
select the AMI instead of using mapping.
Parameters:
   KeyName:
      Description: Name of an existing EC2 KeyPair to enable SSH access to
the instances
      Type: 'AWS::EC2::KeyPair::KeyName'
      ConstraintDescription: must be the name of an existing EC2 KeyPair.
   InstanceType:
      Description: WebServer EC2 instance type
      Type: String
      Default: t2.micro
      AllowedValues:
         - t1.micro
         - t2.micro
         - t2.small
         - t2.medium
         - t2.large
         - t3.micro
         - t3.small
         - t3.medium
         - t3.large
         - m4.large
         - g2.2xlarge
         # - g3.4xlarge
      ConstraintDescription: must be a valid EC2 instance type.

   SSHLocation:
      Description: The IP address range that can be used to SSH to the EC2
instances
      Type: String
      MinLength: '9'
      MaxLength: '18'
      Default: 0.0.0.0/0
      AllowedPattern: '(\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/(\d{1,2})'
      ConstraintDescription: must be a valid IP CIDR range of the form
x.x.x.x/x.
   DBName:
      Default: wordpressdb
      Description: The WordPress database name
      Type: String
      MinLength: '1'
      MaxLength: '64'
      AllowedPattern: '[a-zA-Z][a-zA-Z0-9\-]*'
      ConstraintDescription: must begin with a letter and contain only
alphanumeric characters.
   DBUser:
      NoEcho: 'true'
      Description: The WordPress database admin account username
      Type: String
      MinLength: '1'
      MaxLength: '16'
      AllowedPattern: '[a-zA-Z][a-zA-Z0-9\-]*'
      ConstraintDescription: must begin with a letter and contain only
alphanumeric characters.
   DBPassword:
      NoEcho: 'true'
      Description: The WordPress database admin account password
      Type: String
      MinLength: '8'
      MaxLength: '41'
      AllowedPattern: '[a-zA-Z0-9\-]*'
      ConstraintDescription: must contain only alphanumeric characters.
   DBRootPassword:
      NoEcho: 'true'
      Description: MySQL root password
      Type: String
      MinLength: '8'
      MaxLength: '41'
      AllowedPattern: '[a-zA-Z0-99\-]*'
      ConstraintDescription: must contain only alphanumeric characters.
Mappings:
   AWSInstanceType2Arch:
      t1.micro:
         Arch: HVM64
         # Arch: PV64  need to sort out Linux2 on PV
      t2.nano:
         Arch: HVM64
      t2.micro:
         Arch: HVM64
      t2.small:
         Arch: HVM64
      t2.medium:
         Arch: HVM64
      t2.large:
         Arch: HVM64
      t3.micro:
         Arch: HVM64
      t3.small:
         Arch: HVM64
      t3.medium:
         Arch: HVM64
      t3.large:
         Arch: HVM64
      m1.small:
         Arch: HVM64
      m1.medium:
         Arch: HVM64
      m1.large:
         Arch: HVM64
      m1.xlarge:
         Arch: HVM64
      m2.xlarge:
         Arch: HVM64
      m2.2xlarge:
         Arch: HVM64
      m2.4xlarge:
         Arch: HVM64
      m3.medium:
         Arch: HVM64
      m3.large:
         Arch: HVM64
      m3.xlarge:
         Arch: HVM64
      m3.2xlarge:
         Arch: HVM64
      m4.large:
         Arch: HVM64
      m4.xlarge:
         Arch: HVM64
      m4.2xlarge:
         Arch: HVM64
      m4.4xlarge:
         Arch: HVM64
      m4.10xlarge:
         Arch: HVM64
      c1.medium:
```

```
   Arch: HVM64
c1.xlarge:
   Arch: HVM64
c3.large:
   Arch: HVM64
c3.xlarge:
   Arch: HVM64
c3.2xlarge:
   Arch: HVM64
c3.4xlarge:
   Arch: HVM64
c3.8xlarge:
   Arch: HVM64
c4.large:
   Arch: HVM64
c4.xlarge:
   Arch: HVM64
c4.2xlarge:
   Arch: HVM64
c4.4xlarge:
   Arch: HVM64
c4.8xlarge:
   Arch: HVM64
g2.2xlarge:
   Arch: HVMG2
g2.8xlarge:
   Arch: HVMG2
g3.4xlarge:
   Arch: HVM64
r3.large:
   Arch: HVM64
r3.xlarge:
   Arch: HVM64
r3.2xlarge:
   Arch: HVM64
r3.4xlarge:
   Arch: HVM64
r3.8xlarge:
   Arch: HVM64
i2.xlarge:
   Arch: HVM64
i2.2xlarge:
   Arch: HVM64
i2.4xlarge:
   Arch: HVM64
i2.8xlarge:
   Arch: HVM64
d2.xlarge:
   Arch: HVM64
d2.2xlarge:
   Arch: HVM64
d2.4xlarge:
   Arch: HVM64
d2.8xlarge:
   Arch: HVM64
hi1.4xlarge:
   Arch: HVM64
```

```
      hs1.8xlarge:
         Arch: HVM64
      cr1.8xlarge:
         Arch: HVM64
      cc2.8xlarge:
         Arch: HVM64

Resources:
   AMIInfo:
      Type: Custom::AMIInfo
      Properties:
         ServiceToken:
            Fn::GetAtt:
               - AMIInfoFunction
               - Arn
         Region:
            Ref: AWS::Region
         Architecture:
            Fn::FindInMap:
               - AWSInstanceType2Arch
               - Ref: InstanceType
               - Arch
            # try mapping just on the type and not on the size. !Select [0,
!Split [",", Ref: InstanceTyp]]
   AMIInfoFunction:
      Type: AWS::Lambda::Function
      Properties:
         Code:
            ZipFile: |
               /**
               * A sample Lambda function that looks up the latest AMI ID for
a given region and architecture.
               **/
               // Map instance architectures to an AMI name pattern
               var archToAMINamePattern = {
                   "PV64":  "amzn-ami-pv*x86_64-ebs",
                   "HVM64": "amzn2-ami-hvm-2.0.*x86_64-gp2",
                   "HVMG2": "amzn2-ami-graphics-hvm*x86_64-gp2*"
               //    "HVMG3": "amzn2-ami-graphics-hvm*x86_64-gp2*"
               };
               var aws = require("aws-sdk");
               exports.handler = function(event, context) {
                   console.log("REQUEST RECEIVED:\n" +
JSON.stringify(event));

                   // For Delete requests, immediately send a SUCCESS
response.
                   if (event.RequestType == "Delete") {
                       sendResponse(event, context, "SUCCESS");
                       return;
                   }
                   var responseStatus = "FAILED";
                   var responseData = {};
                   var ec2 = new aws.EC2({region:
event.ResourceProperties.Region});
                   var describeImagesParams = {
                       Filters: [{ Name: "name", Values:
[archToAMINamePattern[event.ResourceProperties.Architecture]]}],
                       Owners: [event.ResourceProperties.Architecture ==
"HVMG2" ? "679593333241" : "amazon"]
                   };
                   // Get AMI IDs with the specified name pattern and owner
                   ec2.describeImages(describeImagesParams, function(err,
describeImagesResult) {
                       if (err) {
                           responseData = {Error: "DescribeImages call
failed"};
                           console.log(responseData.Error + ":\n", err);
                       }
                       else {
                           var images = describeImagesResult.Images;
                           // Sort images by name in descending order. The
names contain the AMI version, formatted as YYYY.MM.Ver.
                           images.sort(function(x, y) { return
y.Name.localeCompare(x.Name); });
                           for (var j = 0; j < images.length; j++) {
                               if (isBeta(images[j].Name)) continue;
                               responseStatus = "SUCCESS";
                               responseData["Id"] = images[j].ImageId;
                               break;
                           }
                       }
                       sendResponse(event, context, responseStatus,
responseData);
                   });
               };
               // Check if the image is a beta or rc image. The Lambda
function won't return any of those images.
               function isBeta(imageName) {
                   return imageName.toLowerCase().indexOf("beta") > -1 ||
imageName.toLowerCase().indexOf(".rc") > -1;
               }
               // Send response to the pre-signed S3 URL
               function sendResponse(event, context, responseStatus,
responseData) {
                   var responseBody = JSON.stringify({
                       Status: responseStatus,
                       Reason: "See the details in CloudWatch Log Stream: " +
context.logStreamName,
                       PhysicalResourceId: context.logStreamName,
                       StackId: event.StackId,
                       RequestId: event.RequestId,
                       LogicalResourceId: event.LogicalResourceId,
                       Data: responseData
                   });
                   console.log("RESPONSE BODY:\n", responseBody);
                   var https = require("https");
                   var url = require("url");
                   var parsedUrl = url.parse(event.ResponseURL);
                   var options = {
                       hostname: parsedUrl.hostname,
                       port: 443,
                       path: parsedUrl.path,
                       method: "PUT",
                       headers: {
                           "content-type": "",
                           "content-length": responseBody.length
                       }
                   };
                   console.log("SENDING RESPONSE...\n");
                   var request = https.request(options, function(response) {
                       console.log("STATUS: " + response.statusCode);
                       console.log("HEADERS: " +
JSON.stringify(response.headers));
                       // Tell AWS Lambda that the function execution is done
                       context.done();
                   });
                   request.on("error", function(error) {
                       console.log("sendResponse Error:" + error);
                       // Tell AWS Lambda that the function execution is done
                       context.done();
                   });
                   // write data to request body
                   request.write(responseBody);
                   request.end();
               }
         Handler:
            index.handler
         Role:
            Fn::GetAtt:
               - LambdaExecutionRole
               - Arn
         Runtime: nodejs10.x
         Timeout: '30'
   LambdaExecutionRole:
      Type: AWS::IAM::Role
      Properties:
         AssumeRolePolicyDocument:
            Version: '2012-10-17'
            Statement:
               -  Effect: Allow
                  Principal:
                     Service:
                        - lambda.amazonaws.com
                  Action:
                     - sts:AssumeRole
         Path: "/"
         Policies:
            -  PolicyName: root
               PolicyDocument:
                  Version: '2012-10-17'
                  Statement:
                     -  Effect: Allow
                        Action:
                           - logs:CreateLogGroup
                           - logs:CreateLogStream
                           - logs:PutLogEvents
                        Resource: arn:aws:logs:*:*:*
                     -  Effect: Allow
                        Action:
                          - ec2:DescribeImages
                       Resource: "*"

   WebServerSecurityGroup:
      Type: 'AWS::EC2::SecurityGroup'
      Properties:
         GroupDescription: >-
            Enable HTTP access via port 80 locked down to the load-balancer +
SSH
            access
         SecurityGroupIngress:
            -  IpProtocol: tcp
               FromPort: '80'
               ToPort: '80'
               CidrIp: 0.0.0.0/0
            -  IpProtocol: tcp
               FromPort: '22'
               ToPort: '22'
               CidrIp: !Ref SSHLocation
   WebServer:
      Type: 'AWS::EC2::Instance'
      Metadata:
         'AWS::CloudFormation::Init':
            configSets:
               wordpress_install:
                  - install_cfn
                  - install_wordpress
                  - configure_wordpress
            install_cfn:
               files:
                  /etc/cfn/cfn-hup.conf:
                     content: !Join
                        - ''
                        -  - |
                              [main]
                           - stack=
                           - !Ref 'AWS::StackId'
                           - |+

                           - region=
                           - !Ref 'AWS::Region'
                           - |+

                     mode: '000400'
                     owner: root
                     group: root
                  /etc/cfn/hooks.d/cfn-auto-reloader.conf:
                     content: !Join
                        - ''
                        -  - |
                              [cfn-auto-reloader-hook]
                           - |
                              triggers=post.update
                           - |

path=Resources.WebServer.Metadata.AWS::CloudFormation::Init
                           - 'action=/opt/aws/bin/cfn-init -v '
                           - '         --stack '
                           - !Ref 'AWS::StackName'
                           - '         --resource WebServer '
                           - '         --configsets wordpress_install '
                           - '         --region '
                           - !Ref 'AWS::Region'
                           - |+

                     mode: '000400'
                     owner: root
                     group: root
               services:
                  sysvinit:
                     cfn-hup:
                        enabled: 'true'
                        ensureRunning: 'true'
                        files:
                           - /etc/cfn/cfn-hup.conf
                           - /etc/cfn/hooks.d/cfn-auto-reloader.conf
            install_wordpress:
               packages:
                  yum:
                     php: []
                     php-mysqlnd: []
                     mysql-community-server: []
                     mysql-community-devel: []
                     mysql-community-client: []
                     mysql-community-libs: []
                     httpd: []
               sources:
                  /var/www/html: 'http://wordpress.org/latest.tar.gz'
               files:
                  /tmp/setup.mysql:
                     content: !Join
                        - ''
                        -  - 'CREATE DATABASE '
                           - !Ref DBName
                           - |
                              ;
                           - CREATE USER '
                           - !Ref DBUser
                           - '''@''localhost'' IDENTIFIED BY '''
                           - !Ref DBPassword
                           - |
                              ';
                           - 'GRANT ALL ON '
                           - !Ref DBName
                           - .* TO '
                           - !Ref DBUser
                           - |
                              '@'localhost';
                           - |
                              FLUSH PRIVILEGES;
                     mode: '000400'
                     owner: root
                     group: root
                  /tmp/create-wp-config:
                     content: !Join
                        - ''
                        -  - |
                              #!/bin/bash -xe
                           - >
                              cp /var/www/html/wordpress/wp-config-sample.php
                              /var/www/html/wordpress/wp-config.php
                           - sed -i "s/'database_name_here'/'
                           - !Ref DBName
                           - |
                              '/g" wp-config.php
                           - sed -i "s/'username_here'/'
                           - !Ref DBUser
                           - |
                              '/g" wp-config.php
                           - sed -i "s/'password_here'/'
                           - !Ref DBPassword
                           - |
                              '/g" wp-config.php
                     mode: '000500'
                     owner: root
                     group: root
               services:
                  sysvinit:
                     httpd:
                        enabled: 'true'
                        ensureRunning: 'true'
                     mysqld:
                        enabled: 'true'
                        ensureRunning: 'true'
            configure_wordpress:
               commands:
                  01_set_mysql_root_password:
                     command: !Join
                        - ''
                        -  - mysqladmin -u root password '
                           - !Ref DBRootPassword
                           - ''''
                     test: !Join
                        - ''
                        -  - '$(mysql '
                           - !Ref DBName
                           - ' -u root --password='''
                           - !Ref DBRootPassword
                           - ''' >/dev/null 2>&1 </dev/null); (( $? != 0 ))'
                  02_create_database:
                     command: !Join
                        - ''
                        -  - mysql -u root --password='
                           - !Ref DBRootPassword
                           - ''' < /tmp/setup.mysql'
                     test: !Join
                        - ''
                        -  - '$(mysql '
                           - !Ref DBName
                           - ' -u root --password='''
                           - !Ref DBRootPassword
```

```
                              - ''' >/dev/null 2>&1 </dev/null); (( $? != 0 ))'
                  03_configure_wordpress:
                     command: /tmp/create-wp-config
                     cwd: /var/www/html/wordpress
                  04_configure_wordpress:
                     command: chown -R apache /var/www/html/wordpress/wp-
content
                  05_configure_wordpress:
                     command: sudo systemctl stop httpd
                  06_configure_wordpress:
                     command: sudo echo "in configure_wordpress" >>
/var/www/html/progress.txt
                  11_configure_wordpress:
                     command: chkconfig httpd on
                  12_configure_wordpress:
                     command:  sudo systemctl start httpd
                  13_configure_wordpress:
                     command:  sudo systemctl enable httpd

      Properties:
         Tags:
            -  Key: Name
               Value: !Ref 'AWS::StackName'
         ImageId:
            Fn::GetAtt:
               - AMIInfo
               - Id

         InstanceType: !Ref InstanceType
         SecurityGroups:
            - !Ref WebServerSecurityGroup
         KeyName: !Ref KeyName
         UserData: !Base64
            'Fn::Join':
               - ''
               -  - |
                     #!/bin/bash -xe
                  - |

                  - |
                     wget http://repo.mysql.com/mysql-community-release-el7-
5.noarch.rpm
                  - |

                  - |
                     rpm -ivh mysql-community-release-el7-5.noarch.rpm
                  - |

                  - |
                     yum update -y
                  - |

                  - |
                     yum install httpd -y
                  - |

                  - |
                     yum install -y amazon-linux-extras
                  - |+

                  - |
                     yum install -y aws-cfn-bootstrap
                  - |+

                  - |
                     echo "in UseData before - amazon-linux-extras enable
php7.3" >> /var/www/html/progress.txt
                  - |+

                  - |
                     amazon-linux-extras enable php7.3
                  - |

                  - |
                     yum clean metadata
                  - |+

                  - |
                     echo "in UseData before - /opt/aws/bin/cfn-init -v" >>
/var/www/html/progress.txt
                  - |+

                  - '/opt/aws/bin/cfn-init -v '
                  - '  --stack '
                  - !Ref 'AWS::StackName'
                  - '  --resource WebServer '
                  - '  --configsets wordpress_install '
                  - '  --region '
                  - !Ref 'AWS::Region'
                  - |+

                  - |+
                     echo "in UseData before - /opt/aws/bin/cfn-signal -e "
>> /var/www/html/progress.txt
                  - |+

                  - '/opt/aws/bin/cfn-signal -e $? '
                  - '   --stack '
                  - !Ref 'AWS::StackName'
                  - '   --resource WebServer '
                  - '   --region '
                  - !Ref 'AWS::Region'
                  - |+

                  - |+
                     echo "in UseData after - /opt/aws/bin/cfn-signal -e" >>
/var/www/html/progress.txt
                  - |+


      CreationPolicy:
         ResourceSignal:
            Timeout: PT15M
Outputs:
   WebsiteURL:
      Value: !Join
         - ''
         -  - 'http://'
            - !GetAtt
               - WebServer
               - PublicDnsName
            - /wordpress
      Description: WordPress Website

   AmiIDused:
      Description: The Amazon EC2 instance AMI ID.
      Value:
         Fn::GetAtt:
            - AMIInfo
            - Id

   MyTypePrefix:
      Description: Type without size.
      Value:
         Fn::Select:
            - 0
            - Fn::Split:
               - "."
               - "Ref": "InstanceType"
```

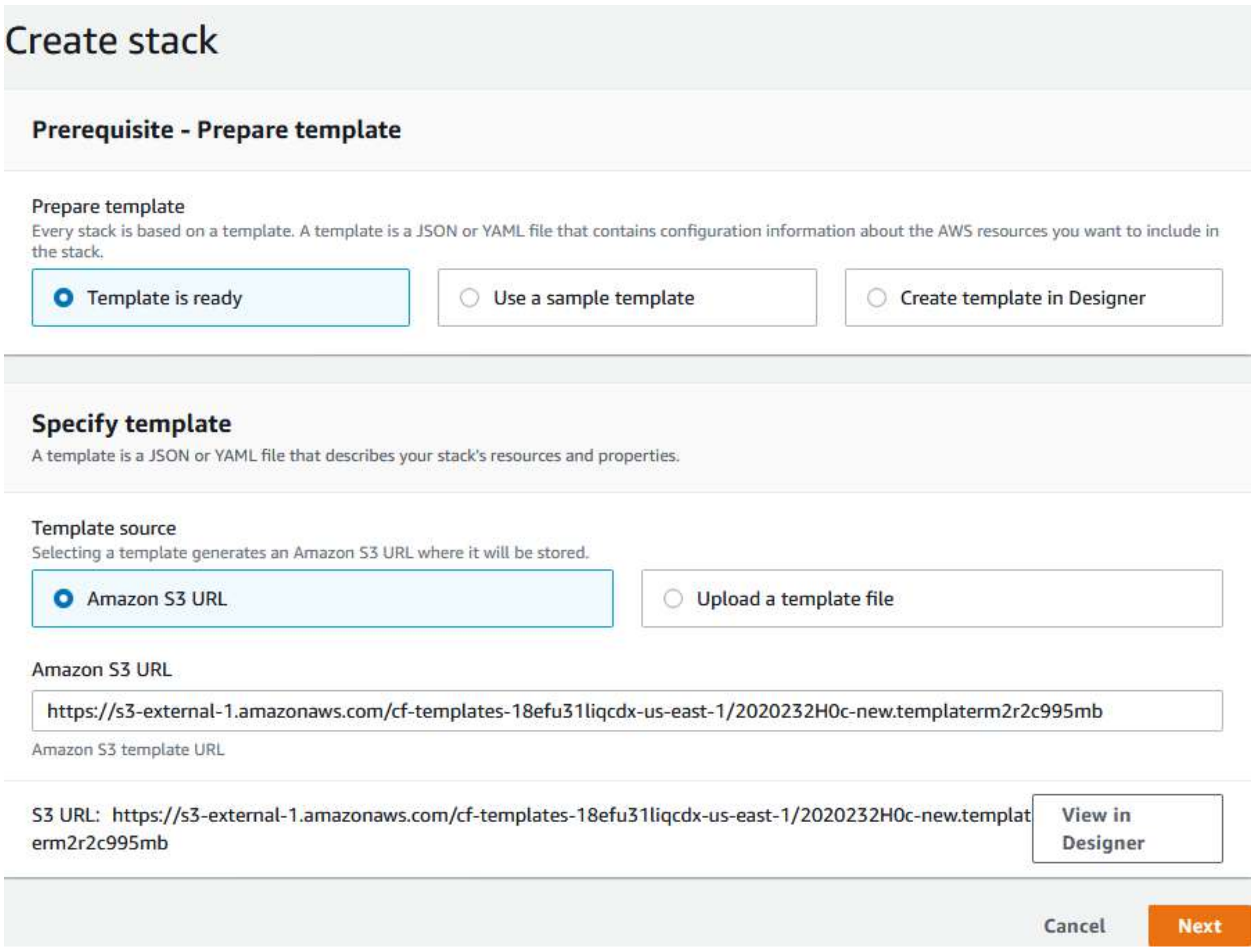
Now that you have pasted in the above code, click the **Refresh Diagram** icon at the top right of the page. This should prompt a diagram of the architecture which consists of a Web Server Security Group, a Web Server instance, AMIinfo, an AMIInfoFunction, and a LambdaExecutionRole. The template installs WordPress with a local MySQL database for storage. Click the **Validate template** icon at the top left of the screen. The response should be: Template is valid. Then click the **Create stack** icon on the top left of the screen.

You will be redirected to the Specify Template/Create stack page.

Prepare template: ✓Template is ready 🡐 will already be selected. Leave it as such.

Template Source: ✓Amazon S3 URL 🡐 will already be selected. Leave it as such.

★See the below screenshot for further clarification as to what the Create Stack page will look like.



Create stack

Prerequisite - Prepare template

Prepare template

Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Template is ready] Use a sample template Create template in Designer

Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source

Selecting a template generates an Amazon S3 URL where it will be stored.

[✓ Amazon S3 URL] Upload a template file

Amazon S3 URL

https://s3-external-1.amazonaws.com/cf-templates-18efu31liqcdx-us-east-1/2020232H0c-new.templaterm2r2c995mb

Amazon S3 template URL

S3 URL: https://s3-external-1.amazonaws.com/cf-templates-18efu31liqcdx-us-east-1/2020232H0c-new.templaterm2r2c995mb

View in Designer

Cancel Next

Click **Next**

This will direct you to the Specify Stack Details page

Stack name: wordpressproject ← For simplicity, give it a name like so

DBName: wordpressproject ← For simplicity, give it a name like so

DBpassword: wordpressproject ← For simplicity, give it a name like so

DBRootPassword: wordpressproject ← For simplicity, give it a name like so

DBUser: wordpressproject ← For simplicity, give it a name like so

InstanceType: ✓t2.micro ←select this

KeyName: ✓WordPressSiteKeyPair ←just select the key pair you created in the previous lesson or select another key pair that you would rather use.

SSHLocation: 0.0.0.0/0 ← Just use 0.0.0.0/0 as we will be deleting the stack soon after we create it.

Click **Next**

This will direct you to the Configure stack options page.

Simply leave everything as default and click **Next**

This will direct you to the Review page.

Scroll down to the bottom of the page and in the Capabilities section check the box that says

✓I acknowledge that AWS CloudFormation might create IAM resources.

Then click the **Create stack** button

Wait a few minutes and then click the **refresh** icon located to the right of **Events**

Once your stack creation is complete, click the **Outputs** tab and then click the **WebsiteURL** link which will look something like the following: http://ec2-3-82-156-45.compute-1.amazonaws.com/wordpress

This will direct you to the WordPress Welcome screen

Site Title: wordpressproject ←type in wordpressproject

Username: wordpressproject ←type in wordpressproject

Password: wordpressproject ←type in wordpressproject

Your email: ← enter your email address

Search Engine Visibility ← leave the box **unchecked**

Click **Install Wordpress**

Then click **Log In**

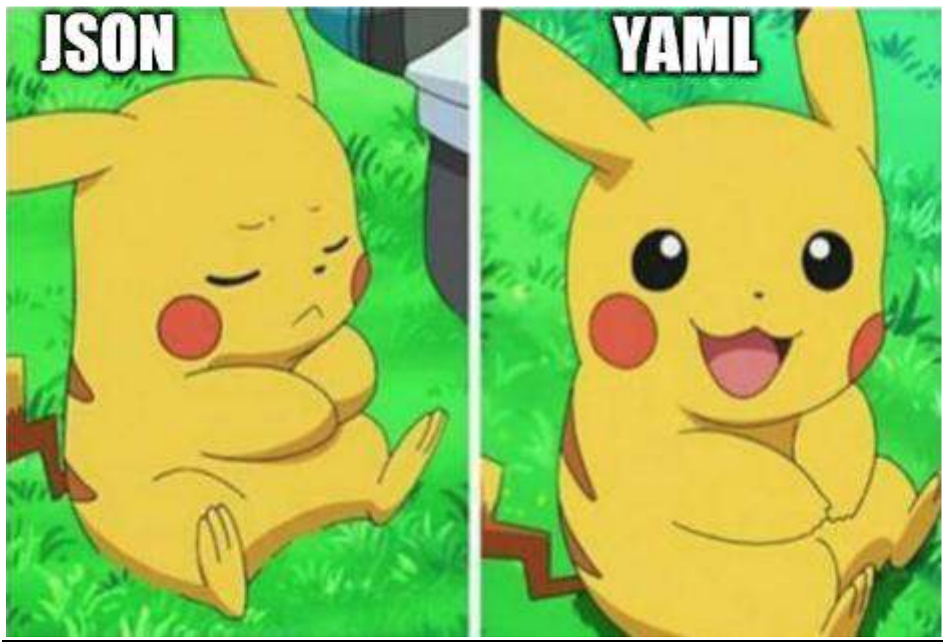
Username: wordpressproject

Password: wordpressproject

Click **Log In**

You are now logged in.

Mission Accomplished!



Now let's clean up.

Return to the AWS console. Click **Services**. Click **CloudFormation.**

✓Select the stack we just created called wordpressproject and then click **Delete**

Then click **Delete stack** to confirm that you wish to delete the stack

Now click **Services** and click **S3** and ✓select the bucket that was automatically created to store our template.

It will have a bucket name similar to: cd-templates-18efu313liqcdx-us-east-1

Once you have selected the bucket click the **delete** button

Then type the name of the bucket (to confirm deletion) and click **Confirm**

As you can see CloudFormation allows you to model your entire infrastructure in a text file. In this project we used YAML to describe what AWS resources to create and configure. And to delete the resources, all we had to do was delete the stack. By the way, there are powerful ready-made templates available in **AWS Quick Starts**. Go check it out when you get a chance.

CloudFormation Cheat Sheet

When being asked to **automate** the provisioning of resources think CloudFormation

-When infrastructure as Code (IaC) is mentioned think CloudFormation

-CloudFormation can be written in either JSON or YAML

-When CloudFormation encounters an error it will rollback with ROLLBACK_IN_PROGRESS

-CloudFormation templates larger than 51,200 bytes (0.05 MB) are too large to upload directly, and must be imported into CloudFormation via an S3 bucket.

-NestedStacks helps you break up your CloudFormation template into smaller reusable templates that can be composed into larger templates

-At least one resource under resources: must be defined for a CloudFormation template to be valid

-MetaData extra information about your template

-Description a description of what the template is supposed to do

-Parameters is how you get user inputs into templates

-Transforms Applies macros (like applying a mod which change the anatomy to be custom)

-Outputs are values you can use to import into other stacks

-Mappings maps keys to values, just like a lookup table

-Resources defines the resources you want to provision, at least one resource is required

-Conditions are whether resources are created or propertiets are assigned

---

Question: A team is designing the architecture for a new application with full CI/CD testing. They want to implement feature branch testing based on pull requests to master. A Pull Request should cause a full deployment to be run on that feature branch being pulled so that a tester can run through functional tests. What would you recommend the team does to automate this process at the lowest cost?

A) Use Amazon EC2 Reserved Instance and Amazon CloudFormation to deploy a testing environment at lowest cost
B) Use Amazon EC2 Spot Fleet and Amazon CloudFormation to deploy an integration testing environment at lowest cost
C) Configure CloudWatch Events to trigger a deployment based on pull requests
D) Configure AWS CloudTrail to log pull request events and trigger a deployment

---

Explanation:
CloudFormation allows AWS to automatically deploy the infrastructure required to deploy the application for testing. The infrastructure code can be stored alongside application code to allow the application to be deployed in a fully-isolated infrastructure which can be destroyed once integration testing is complete. CloudWatch Events (and not CloudTrail) enables pull request events to trigger a deployment. Finally Amazon EC2 Spot Fleet allow us to deploy a set of EC2 instances at the lowest cost. Reserved Instances are better-suited to pre-purchasing compute capacity which you will use for a fixed period of time - it is not cost-effective to pre-purchase EC2 capacity just to perform integration testing.
Resources. Answer: B & C