云享家

3 步轻松利用 CloudFormation 搭建云上环境

BESPIN GLOBAL

创建一台 EC2 的过程，包括选择镜像、选择规格、配置网络、配置存储，配置安全组、配置密钥对、打标签等。创建一次还好，但是如果要创建十次，就有点头晕了。设想要在短时间内构建出上百套环境，要求每个环境有多台虚拟机，环境之间可以不同且需要支持自定义。比如我们公司的一个项目，为温州大学网络工程的学生提供上实验课用的仿真环境，就不可能通过手工操作来做到，而这恰恰是 CloudFormation 的用武之地。

通过编写 CloudFormation，可以将基础设施仅仅视为代码， 使用任何代码编辑器对它进行修改，也可以并入版本控制体系中，与团队成员一起查看文件，然后很便捷地构建出真实环境。

通过阅读本文，您将能做到：

1. 创建一台 Linux 虚拟机，熟悉模板基础语法
2. 创建一台可直接使用账号密码登录的 Linux 虚拟机、Windows 虚拟机，免去首次登录强制使用密钥对的烦恼，熟悉在两种操作系统下的模板初始化
3. 用编程的方式创建、删除、查看堆栈，熟悉使用 Java SDK 调用 CloudFormation 的 API

1. CloudFormation 创建虚拟机

1.1. CloudFormation 基础语法：

如下所示，是创建一台 EC2 的完整 CloudFormation 模板。通过一个最典型的例子，为大家介绍模板的基础语法。也可以访问 https://bp-cloudshare.s3.cn-northwest-1.amazonaws.com.cn/cloudshare-linux-ec2.template 来获取模板源码。

```
{
 "AWSTemplateFormatVersion" : "2010-09-09",

 "Description" : "AWS CloudFormation Linux EC2 Template.",

 "Parameters" : {
  "KeyName": {
   "Description" : "Name of an existing EC2 KeyPair to enable SSH access to the instance.",
   "Type": "AWS::EC2::KeyPair::KeyName",
   "ConstraintDescription" : "must be the name of an existing EC2 KeyPair."
  },
  "InstanceType" : {
   "Description" : "EC2 instance type",
   "Type" : "String",
   "Default" : "t2.micro",
   "AllowedValues" : [ "t1.micro", "t2.nano", "t2.micro", "t2.small"],
   "ConstraintDescription" : "must be a valid EC2 instance type."
  }
 },

 "Resources" : {
  "LinuxInstance" : {
   "Type" : "AWS::EC2::Instance",
   "Properties" : {
    "InstanceType" : { "Ref" : "InstanceType" },
    "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],
    "KeyName" : { "Ref" : "KeyName" },
    "ImageId" : "ami-094b7433620966eb5",
    "Tags": [
     {
      "Key": "Name",
      "Value": "linux-sample"
     }
    ]
   }
  },
  "InstanceSecurityGroup" : {
   "Type" : "AWS::EC2::SecurityGroup",
   "Properties" : {
    "GroupDescription" : "Enable SSH access via port 22",
    "SecurityGroupIngress" : [ {
     "IpProtocol" : "tcp",
     "FromPort" : "22",
     "ToPort" : "22" ,
          "CidrIp" : "0.0.0.0/0"
    } ]
```

```
47.      }
48.    }
49.  },
50.
51.  "Outputs" : {
52.    "link" : {
53.      "Description" : "the description of link",
54.      "Value" : {
55.        "Fn::Join": [",", [{"Fn::GetAtt":["LinuxInstance", "PublicIp"]},
56.          {"Fn::GetAtt": ["LinuxInstance", "PrivateIp"]}]]
57.      }
58.    }
59.  }
60. }
```

第 2 行的 **AWSTemplateFormatVersion** 是 CloudFormation 的版本号，因为迄今为止只有一个版本 2010-09-09，所以这一项可以当成是固定值。

第 4 行的 **Description** 是 CloudFormation 的描述，可以是任何文本信息，比如描述这个模板的用途、有哪些注意事项、生成了怎样的环境等，类似于注释。

第 6 行的 **Parameters** 是 CloudFormation 的参数部分，利用参数，每次创建或更新堆栈时可以将自定义值输入模板，然后在 Resources 和 Outputs 部分中，可以使用 Ref 内部函数来引用某个参数。上面的例子定义了两个参数，分别是密钥对名称 KeyName 和实例规格 InstanceType。在参数内部，Type 属性是参数类型，Default 属性是默认值，AllowedValues 属性是允许的值。

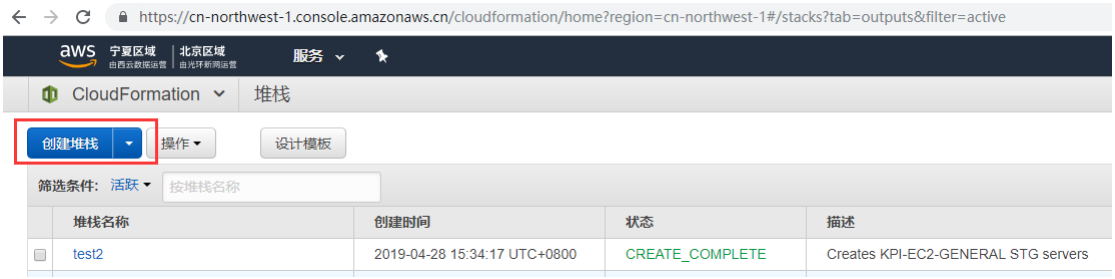
第 21 行的 **Resources** 是 CloudFormation 的资源部分，是模板最重要的一个部分，也是唯一必需的部分。它用来声明包含在堆栈中的 AWS 资源，例如 EC2 实例或 S3 存储桶。上面的例子定义了两种资源，分别是 EC2 实例 LinuxInstance 和安全组 InstanceSecurityGroup，两者通过 EC2 实例的 SecurityGroups 属性关联起来。在资源内部，Type 属性是资源类型，Properties 属性定义了资源的详细信息。

第 51 行的 **Outputs** 是 CloudFormation 的输出部分，用来在控制台或者 API 调用中返回自定义内容，比如输出堆栈的 S3 存储桶名称以使该存储桶更容易找到。上面的例子输出了 EC2 实例的公网 IP 和内网 IP，其中的 Fn::Join 和 Fn::GetAtt 是 CloudFormation 的内部函数，分别用来拼接字符串和获取资源属性。

1.2. CloudFormation 控制台操作：

有了模板之后，就可以通过 AWS CloudFormation 控制台把它变为基础设施。这个转变是非常方便的，下面为大家介绍控制台的操作：

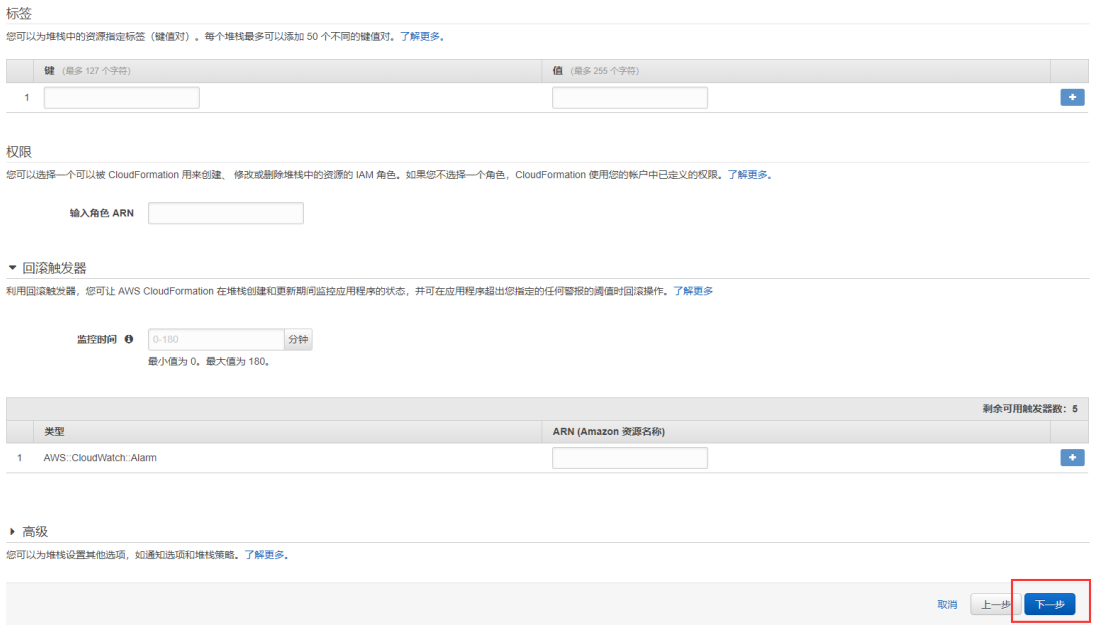
1) 进入 CloudFormation 控制台，点击创建堆栈



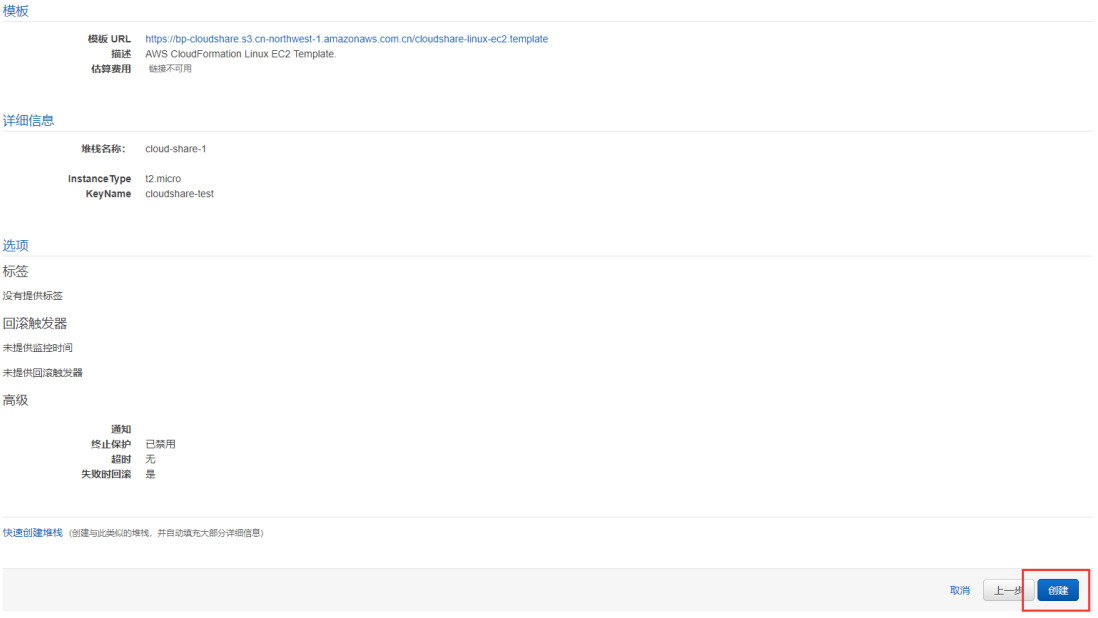
2) 选择模板，指定 S3 模板 URL，点击下一步

3) 输入堆栈名称，给参数赋值，点击下一步

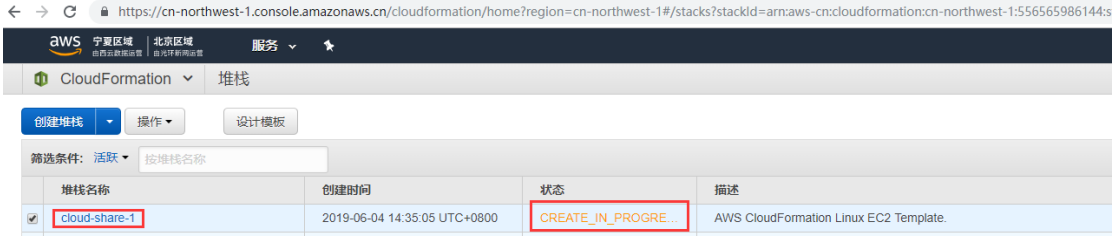
4) 选项页，可以不配置，点击下一步



5) 审核页，审核通过后，点创建



6) 创建后，回到列表页，可以看见刚才创建的堆栈。当前状态为创建中，稍后状态会变为创建成功。



2. CloudFormation 初始化

2.1. Linux 初始化：

众所周知，使用 AWS 公共 Linux 镜像创建的虚拟机，首次登录的时候，只能通过密钥对的方式。这是 AWS 强制的，在控制台并没有预配置账号密码的地方，只能用密钥对登录后再用脚本重置 root 账号密码。有些时候我们并不愿意这样做，我们更想直接使用账号密码来登录虚拟机，比如临时环境并不需要太高的安全性、私有密钥容易忘记存放位置、客户要求用账号密码登录等。通过解决这个有意思的问题，为大家介绍 CloudFormation 的 Linux 初始化，完整的源码可以访问 https://bp-cloudshare.s3.cn-northwest-1.amazonaws.com.cn/cloudshare-password-login-linux-ec2.template 获取。

```
1.  "Resources" : {
2.    "LinuxInstance" : {
3.     "Type" : "AWS::EC2::Instance",
4.     "Metadata": {
5.      "AWS::CloudFormation::Init": {
6.       "configSets": {
7.        "ConfigRoot": ["Configure"]
8.       },
9.       "Configure": {
10.       "commands": {
11.        "01_set_root_password": {
12.         "command": {
13.          "Fn::Join": ["", ["echo '", {
14.           "Ref": "Password"
15.          }, "' | sudo passwd --stdin root"]]
16.         }
17.        },
18.        "02_set_root_password": {
19.         "command": "sudo sed -
    i 's/PasswordAuthentication no/PasswordAuthentication yes/g' /etc/ssh/sshd_config"
20.        },
21.        "03_set_root_password": {
22.         "command": "sudo /sbin/service sshd restart"
23.        }
24.       }
25.      }
26.     }
27.    },
28.    "Properties" : {
29.     "InstanceType" : { "Ref" : "InstanceType" },
30.     "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],
31.     "KeyName" : { "Ref" : "KeyName" },
32.     "ImageId" : "ami-094b7433620966eb5",
33.     "Tags": [
34.      {
35.       "Key": "Name",
36.       "Value": "linux-sample"
37.      }
38.     ],
39.     "UserData": {
40.      "Fn::Base64": {
41.       "Fn::Join": ["", [
42.        "#!/bin/bash -xe\n",
43.        "yum install -y aws-cfn-bootstrap\n",
44.        "/opt/aws/bin/cfn-init -v ",
45.        "       --stack ", {
46.         "Ref": "AWS::StackName"
47.        },
48.        "       --resource LinuxInstance ",
49.        "       --configsets ConfigRoot ",
50.        "       --region ", {
51.         "Ref": "AWS::Region"
52.        }, "\n"
53.       ]]
54.      }
55.     }
```

```
56.    }
57.   },
```

使用类型 AWS::CloudFormation::Init 来将 Amazon EC2 实例上的元数据置入 cfn-init 帮助程序脚本。当模板调用 cfn-init 脚本时，该脚本会查询来源于 AWS::CloudFormation::Init 元数据键内的资源元数据。

从第 4 行到第 22 行是核心代码，定义了初始化脚本，用这段脚本来初始化服务器，实现使用账号密码直接登录 EC2，而不必通过密钥对来首次登录。脚本主要有三行：

1. echo ‘your_password’ | sudo passwd --stdin root

   解释：--stdin，这个选项用于从标准输入管道读入新的密码，也支持管道的方式；使用 echo 方式 来重置 root 用户密码

2. sudo sed -i 's/PasswordAuthentication no/PasswordAuthentication yes/g' /etc/ssh/sshd_config

   解释：sed -i 's/原字符串/新字符串/g'，用来处理文本文件（/etc/ssh/sshd_config），把“PasswordAuthentication no”替换成“PasswordAuthentication yes”

3. sudo /sbin/service sshd restart

   解释：重启 sshd 服务

这三行脚本，先重置 root 用户密码，再设置服务器允许密码验证登录，最后重启服务使其生效。

从 39 行到 52 行演示了 EC2 实例的 UserData 属性，该属性的作用主要是配置 cfn-init 帮助程序脚本。它读取来自 AWS::CloudFormation::Init 的元数据并进行相应操作,写法较为固定，有—stack（堆栈名称）、--resource（资源名称）、--configsets（配置集）、--region（地域）等配置项，根据实际情况填入相应的值即可。

2.2. Windows 初始化

在 AWS 上用 Windows 公共镜像创建虚拟机后，需要先用密钥对获取登录密码，才能登录服务器，而且自动生成的密码是随机的，基本不可能记住，所以一般还要多一个重置密码的步骤，这样下来就比较繁琐了。通过 CloudFormaiton 的 Windows 初始化，可以预设密码，直接使用密码登录，免去了使用密钥对获取密码和重置密码的步骤。下面通过模板实现这个功能，完整的源码可以访问 https://bp-cloudshare.s3.cn-northwest-1.amazonaws.com.cn/cloudshare-password-login-windows-ec2.template 获取。

```
1.  "Resources" : {
2.      "WindowsInstance" : {
3.        "Type" : "AWS::EC2::Instance",
4.        "Metadata" : {
5.          "AWS::CloudFormation::Init" : {
6.            "config" : {
7.              "files" : {
8.                "c:\\cfn\\cfn-hup.conf" : {
9.                  "content" : { "Fn::Join" : ["", [
10.                   "[main]\n",
11.                   "stack=", { "Ref" : "AWS::StackId" }, "\n",
12.                   "region=", { "Ref" : "AWS::Region" }, "\n"
13.                 ]]}
14.               },
15.               "c:\\cfn\\hooks.d\\cfn-auto-reloader.conf" : {
16.                 "content": { "Fn::Join" : ["", [
17.                   "[cfn-auto-reloader-hook]\n",
18.                   "triggers=post.update\n",
19.                   "path=Resources.WindowsInstance.Metadata.AWS::CloudFormation::Init\n",
20.                   "action=cfn-init.exe -v -s ", { "Ref" : "AWS::StackId" },
21.                   " -r WindowsInstance",
22.                   " --region ", { "Ref" : "AWS::Region" }, "\n"
23.                 ]]}
24.               }
25.             },
26.             "commands" : {
27.               "1-init-user" : {
28.                 "command" : { "Fn::Join" : [ "", ["C:\\Windows\\System32\\net.exe user Administrator ",
29.                   {"Ref": "Password"}]]
30.                 }
31.               },
32.               "2-signal-success" : {
33.                 "command" : { "Fn::Join" : [ "", [
34.                   "cfn-signal.exe -e %ERRORLEVEL% \"",
```

```
35.                   { "Fn::Base64" : { "Ref" : "WindowsServerWaitHandle" }},
36.                   "\""]]
37.                 }
38.               }
39.             },
40.             "services" : {
41.               "windows" : {
42.                 "cfn-hup" : {
43.                   "enabled" : "true",
44.                   "ensureRunning" : "true",
45.                   "files" : ["c:\\cfn\\cfn-hup.conf", "c:\\cfn\\hooks.d\\cfn-auto-
    reloader.conf"]
46.                 }
47.               }
48.             }
49.           }
50.         }
51.       },
52.       "Properties" : {
53.         "InstanceType" : { "Ref" : "InstanceType" },
54.         "SecurityGroups" : [ { "Ref" : "InstanceSecurityGroup" } ],
55.         "KeyName" : { "Ref" : "KeyName" },
56.         "ImageId" : "ami-0a3304f22574e61a8",
57.         "UserData" : { "Fn::Base64" : { "Fn::Join" : ["", [
58.           "<script>\n",
59.           "cfn-init.exe -v -s ", { "Ref" : "AWS::StackId" },
60.           " -r WindowsInstance",
61.           " --region ", { "Ref" : "AWS::Region" }, "\n",
62.           "</script>"
63.         ]]}},
64.         "Tags": [
65.           {
66.             "Key": "Name",
67.             "Value": "windows-sample"
68.           }
69.         ]
70.       }
71.     },
```

第 8 行和第 15 行创建了两个文件，都置于服务器上的 C:\cfn 目录中。它们是：

- cfn-hup.conf，为 cfn-hup 的配置文件。
- cfn-auto-reloader.conf，为挂钩的配置文件，当 AWS::CloudFormation::Init 中的元数据发生变化时，cfn-hup 将用其初始化更新（调用 cfn-init）。

由于实例中的命令可通过按名称的字母顺序排列的方式来处理，因此，每项命令均将预置数字，表示其所需的执行顺序。

在该部分中，UserData 属性包含 cfn-init 执行的 cmd.exe 脚本，并放在<script>标签内。也可以通过将脚本用<powershell>标签括起，来在此处改用 Windows Powershell 脚本。对于 Windows 堆栈，必须再次对等待条件句柄 URL 进行 base64 编码。

Windows 初始化的代码量略多，虽然必不可少但是大部分较为固定，可以直接复用。“commands”属性定义了初始化脚本，“1-init-user”用来重置密码，“2-signal-success”用来在重置密码成功后向 CloudFormation 发送成功信号。

3. CloudFormation 接口调用

用控制台调用 CloudFormation 不能满足所有需求，当需要在自己的系统上使用 CloudFormation，或者通过程序实现更高程度的自动化时，就需要调用 CloudFormaiotn 对外开放的接口。一般有两种方式，使用 Rest API 或者使用 SDK。前者比较轻量，后者比较快捷，可以说后者是对前者的封装。下面为大家介绍 CloudFormation Java SDK 的使用。

3.1. 引入依赖

```xml
<!-- https://mvnrepository.com/artifact/com.amazonaws/aws-java-sdk-cloudformation -->
<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-cloudformation</artifactId>
    <version>1.11.440</version>
</dependency>
```

3.2. 编写帮助类

```java
@Component
public class CfnHelper {
    @Value("${aws.cfn.region}")
    private String region;
    @Value("${aws.cfn.accessKey}")
    private String accessKey;
    @Value("${aws.cfn.serectKey}")
    private String serectKey;
    public static final String STACK_CREATE_COMPLETE = "CREATE_COMPLETE";
    public static final String STACK_DELETE_COMPLETE = "DELETE_COMPLETE";
    private static AmazonCloudFormation client = null;
    public static AmazonCloudFormation getClient() {
        return client;
    }

    @PostConstruct
    public void init() {
        AWSCredentials credentials = new BasicAWSCredentials(accessKey,serectKey);
        client = AmazonCloudFormationClientBuilder
                .standard().withCredentials(new AWSStaticCredentialsProvider(credentials))
                .withRegion(region)
                .build();
    }
}
```

3.3. 创建堆栈

```java
Parameter parameterKeyName = new Parameter()
        .withParameterKey("KeyName")
        .withParameterValue(keyName);
CreateStackRequest request = new CreateStackRequest()
        .withStackName(stackName).withTemplateURL(templateUrl)
        .withParameters(parameterKeyName);
CreateStackResult result = CfnHelper.getClient().createStack(request);
```

3.4. 删除堆栈

```java
DeleteStackRequest request = new DeleteStackRequest().withStackName(stackId);
CfnHelper.getClient().deleteStack(request);
```

3.5. 查询堆栈状态

```java
@Component
@EnableScheduling
public class ScheduledService {
    /**
     * 每隔 10 秒查询一次堆栈创建状态,
     */
    @Scheduled(fixedDelay = 10000)
    public void scheduled_create_stack() {
        //示例代码, stacks当作是多个刚创建的堆栈
        List<TechStackEntity> stacks = new ArrayList<>();

        for(TechStackEntity stackEntity : stacks){
            DescribeStacksRequest request = new DescribeStacksRequest().withStackName(stackEntity.getStackId());
            DescribeStacksResult result = CfnHelper.getClient().describeStacks(request);
            Stack stack = result.getStacks().get(0);
            if(stack.getStackStatus().equals(CfnHelper.STACK_CREATE_COMPLETE) ){
                //TODO 更新数据库堆栈状态
            }
        }
    }
}
```