PaaS on AWS: how to build it the perfect way – Part II

14 October 2022 - 6 min. read

Amazon EC2 | Amazon RDS | Amazon S3 | AWS CodeBuild | AWS CodeDeploy | AWS CodePipeline | CI/CD | GitLab | Platform-as-a-Service (PaaS) | Virtual Host

Welcome back to our 3-step blog post series about building PaaS on AWS the correct way. In Part I, we analyzed the key points for the correct implementation of a PaaS product.

In this second episode, we are creating a **Web Server vending machine** while examining the common **infrastructure stack** for each customer. If you are new to this series, we suggest starting from here, as we are referring to the features and aspects mentioned in part I.

In the repository that we will analyze we find the stacks needed for creating the following:

- Services for interception of **pushes on GitLab**:
  - API Gateway to accept GitLab webhook calls;
  - Lambda to create a configuration file with committed data;
  - CodeBuild Job to pull the repository and upload to S3.

- Dedicated **IAM roles** per environment:
  - Role for the use of the infrastructural pipeline;
  - Instance profile for EC2 instances

  - Role for the use of the software pipeline
  - Role for deploying resources related to EC2 instances, such as AutoScaling Group, Application Load Balancer, etc.

- VPC per environment:
  - CIDR /16
  - 9 subnet:
    - 3 Private
    - 3 Natted
    - 3 Public

  - NAT Instance

- KMS key for the encryption of every object and service for the environment
- **Amazon S3 bucket** for the management of the files used by the pipelines (for example the artifacts) and for the collection of logs divided for each environment
- **Application load balancer** per environment:
  - Listener on 80 port with automatic redirect on 443 with HTTPS protocol
  - Listener on port 443 with return of the 503 error in case of non-match of the present rules

VPC

The VPC consists of **9 subnets** - 3 for each Availability Zone - in order to make the infrastructure **highly available**. They are divided into:

- PUBLIC. Used for all services that must be reached from the internet (such as the ALB) or in case you need to directly expose an EC2 instance by assigning it a dedicated public IP address;
- NATTED. Used for all services that need access to the internet but which must **not be reachable** from the outside; as the name suggests, the instances that will be created within these subnets will be able to access the internet through NAT gateways placed in their public subnets. In our case, we chose to opt for the 3

instances (one for AZ) only for the production VPC, while we're using only one instance for the other environments;

- PRIVATE. Used for all services that do not require internet access such as the RDS database.

With the VPC construct that AWS CDK makes available it is impossible to perform supernetting, since it has a management of the CIDRs assigned to the subnets. This deprives us of the possibility of grouping subnets with smaller netmasks. Therefore, we decided to use this construct, but making sure to **overwrite the various CIDRs** before deploying through this piece of code:

```
myVpc.privateSubnets.forEach((subnet, index) => {
let cidr = `${startSubnetsCidr}.${firstPrivateCidr + index}.${endSubnetsCidr}`
const cfnSubnet = subnet.node.defaultChild as aws_ec2.CfnSubnet;
cfnSubnet.addPropertyOverride('CidrBlock', `${cidr}`);
let name =  `${configFile.projectName}-${process.env.ENVIRONMENT}-natted-${subnet.availabilityZone.replace(/^\w+\-\w+\-\d/,'')}`;
let subName =  `Subnet-Natted-${subnet.availabilityZone.replace(/^\w+\-\w+\-\d/,'').toUpperCase()}-${process.env.ENVIRONMENT}-Name`;
let subId =  `Subnet-Natted-${subnet.availabilityZone.replace(/^\w+\-\w+\-\d/,'').toUpperCase()}-${process.env.ENVIRONMENT}-ID`;
let subCidr =  `Subnet-Natted-${subnet.availabilityZone.replace(/^\w+\-\w+\-\d/,'').toUpperCase()}-${process.env.ENVIRONMENT}-CIDR`;
cdk.Aspects.of(subnet).add(
new cdk.Tag(
'Name',
Name
)
)
})
```

Once the VPC has been deployed, we can deploy all the resources necessary to operate the vending machine such as the **Application Load Balancers** in the public subnets, the **Web Servers** in the natted subnets, and the **databases** dedicated to the Web Servers in the private subnets.

The creation of these resources will be the subject of our next article.

Amazon S3 Bucket

The S3 bucket created by this stack is used to store logs, artifacts and the result of git pushes on GitLab; In addition, relative permissions are assigned for IAM roles, guaranteeing full access to the bucket, and the removal policies for the stored logs are created:

```
const myLifeCycleLogsRule: aws_s3.LifecycleRule = {
        id: `logs-cleared`,
enabled: true,
prefix: `*-${process.env.ENVIRONMENT}-log`,
expiration: cdk.Duration.days(1)
}
```

In order to use the S3 bucket as a pipeline source, the CloudTrail service must be activated to ensure the ability to intercept events:

```
const myTrail = new aws_cloudtrail.Trail(this, `CloudTrail-${process.
env.ENVIRONMENT}`, {
trailName: `trail-${process.env.ENVIRONMENT}`,
sendToCloudWatchLogs: true,
bucket: myGlobalBucketS3,
encryptionKey: myKms,
cloudWatchLogGroup: new aws_logs.LogGroup(this, `Logs-${upperEnvironm
ent}`, {
logGroupName: `logs-${process.env.ENVIRONMENT}`,
retention: aws_logs.RetentionDays.THREE_DAYS,
removalPolicy: RemovalPolicy.DESTROY
}),
cloudWatchLogsRetention: aws_logs.RetentionDays.THREE_DAYS,
s3KeyPrefix: `logs-${process.env.ENVIRONMENT}`,
isMultiRegionTrail: false
});
```

But this is not enough.

To ensure that the pipeline is invoked when a new file is inserted into the S3 bucket, it is necessary to configure **a notification event on CloudTrail** that listens for *write* operations within the S3 bucket:

```
myTrail.addS3EventSelector([{
        bucket: myGlobalBucketS3,
        objectPrefix: `software/`,
        }], {
        readWriteType: aws_cloudtrail.ReadWriteType.WRITE_ONLY,
})
myTrail.addS3EventSelector([{
        bucket: myGlobalBucketS3,
        objectPrefix: `infrastructure/`,
        }], {
        readWriteType: aws_cloudtrail.ReadWriteType.WRITE_ONLY,
})
```

KMS Key

To ensure data encryption on S3, CloudTrail, and in the database, we have created a customer-managed KMS key. We have subsequently assigned a policy to this key that allows entities that must operate on encrypted services to be able to use it:

```
myKms.addToResourcePolicy( new iam.PolicyStatement({
sid: "Allow principals in the account to decrypt log files",
actions: [
"kms:Decrypt",
"kms:ReEncryptFrom"
],
principals: [ new iam.AccountPrincipal(`${process.env.CDK_DEFAULT_ACC
OUNT}`) ],
resources: [
`arn:aws:kms:${process.env.CDK_DEFAULT_REGION}:${process.env.CDK_DEFA
ULT_ACCOUNT}:key/*`,
],
conditions: {
```

```
"StringLike": {
"kms:EncryptionContext:aws:cloudtrail:arn": "arn:aws:cloudtrail:*:*:t
rail/*"
},
"StringEquals": {
"kms:CallerAccount": `${process.env.CDK_DEFAULT_ACCOUNT}`
}
}
}));
```

Application Load Balancer

This ALB will manage the access to our services by automatically directing them from port 80 in HTTP to port 443 in HTTPS:

```
myAppLoadBalancer.addListener(`App-80-Listener`, {
port: 80,
defaultAction: elbv2.ListenerAction.redirect({
permanent: true,
port: '443',
protocol: 'HTTPS',
})
})
myAppLoadBalancer.addListener(`App-443-Listener`, {
port: 443,
defaultAction: elbv2.ListenerAction.fixedResponse(503, {
contentType: `text/plain`,
messageBody: 'host not found',
})
})
```

To manage the requests made in HTTPS on port 443, a certificate must be associated with the relative listener. We can do this using AWS Certificate Manager. This service makes it easy to create and configure certificates allowing also automatic updating.

To conclude

The resources configured within this repository can be considered the foundation for the entire solution.

In the next episode, we will analyze the application stack dedicated to each customer who uses the services we have seen today.

To have a solid solution from the security and scalability point of view, reliability must be firstly ensured to the underlying infrastructure. For this reason, we have relied solely on services managed by AWS, thus reducing the effort of administration and monitoring.

Is everything running smoothly till now?

At this point, we are ready to create the resources. But for this last step see you in 14 days with the last!

About Proud2beCloud

Proud2beCloud is a blog by beSharp, an Italian APN Premier Consulting Partner expert in designing, implementing, and managing complex Cloud infrastructures and advanced services on AWS. Before being writers, we are Cloud Experts working daily with AWS services since 2007. We are hungry readers, innovative builders, and gem-seekers. On Proud2beCloud, we regularly share our best AWS pro tips, configuration insights, in-depth news, tips&tricks, how-tos, and many other resources. Take part in the discussion!

Antonio Callegari

DevOps Engineer @ beSharp.Born as a hardware-addicted, “classic” system engineer, I also like jumping to the dark side: the Cloud! And you know the effect that this mix can make :) Hand-making is my first choice, but a bit of high-quality automation is welcome in my projects.My free time is split between my family and the music, both as a player, and sound engineer.

Mattia Costamagna

DevOps engineer and cloud-native developer @ beSharp. I love spending my free time reading novels and listening to 70s rock and blues music. Always in search of new technologies and frameworks to test and use. Craft beer is my fuel!