AUGUST 2020

Deploy your own production-ready Jenkins in AWS ECS

BY TOM GREGORY

Hosting

Deploying a continuous integration service such as Jenkins is an important step when kicking off your development project.

In this article you'll discover how to deploy Jenkins into the AWS Elastic Container Service (ECS), meaning you'll have your own highly available Jenkins instance available to you over the internet.

We'll be following all the best practices to get Jenkins production-ready, including making sure:

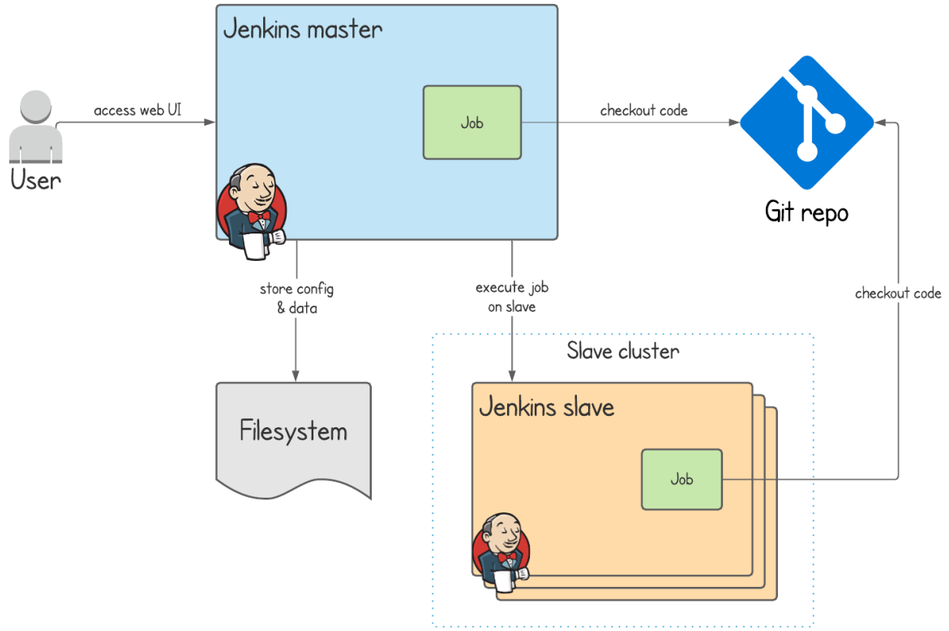
- Jenkins is always available to you even if an entire AWS availability zone goes down all Jenkins data is stored on a persistent volume
- Jenkins runs inside a private network with strict security controls to ensure nobody except
- authenticated users have access
- any changes to the infrastructure can be made easily via AWS CloudFormation templates

This wish list is not difficult to achieve when using services from a cloud provider such as AWS. First though. let's get an understanding of how Jenkins works before figuring out how to deploy it to AWS.

This is the first article in this three-part series about deploying Jenkins into AWS. Here are details of all three articles:

- in **Part 1** *Deploy your own production-ready Jenkins in AWS ECS* (this article) we'll explore how to setup a robust Jenkins master in AWS using CloudFormation
- in **Part 2** *Running Jenkins jobs in AWS ECS with slave agents* we'll get slave jobs running in ECS through a full worked example, doing all the cloud configuration manually for a full understanding of the process
- in **Part 3** *Using Jenkins Configuration as Code to setup AWS slave agents* we'll improve what we had in part 2 by setting up our Jenkins master's cloud configuration automatically using Jenkins Configuration as Code

1. JENKINS ARCHITECTURE OVERVIEW



This diagram describes a common Jenkins use case, where we need to run jobs that build software whose code is checked out of a version control system such as Git.

Jenkins is designed to run a **single master node** responsible for serving the web UI, handling configuration, running jobs, and managing interactions with slave nodes. **Jenkins slave nodes** also run jobs, and they allow for horizontal scalability since you can have many resource intensive jobs running at the same time without affecting the master node.

Also bear in mind that:

- Jenkins stores it's configuration and data (workspaces, job history etc.) on a filesystem accessed from the master node
- because of this, only a single master node can run at the same time otherwise there is risk of data corruption
- using a single master node without slaves is fine for light Jenkins usage

To keep things focused, for this article we'll aim to deploy a single master node for running jobs. Executing jobs on slave nodes will be covered in a follow-up article.

2. WHY WOULD YOU DEPLOY JENKINS INTO ECS?

AWS ECS is a **container orchestration framework** much like its better-known more fashionable cousin Kubernetes. ECS provides everything you need to deploy services as Docker containers, including handing scaling, failover, networking, and security.

The other nice things about ECS are:

- it's fairly straightforward to get up and running quickly, especially if you use a templating language like **CloudFormation** or **Terraform**
- it integrates very well with other AWS technologies such as **Application Load Balancers**, **Security Groups**, and **EC2**

If you know that AWS is the cloud provider you want to use in the long-term, then for getting a service such as Jenkins deployed ECS is an ideal choice. 👍

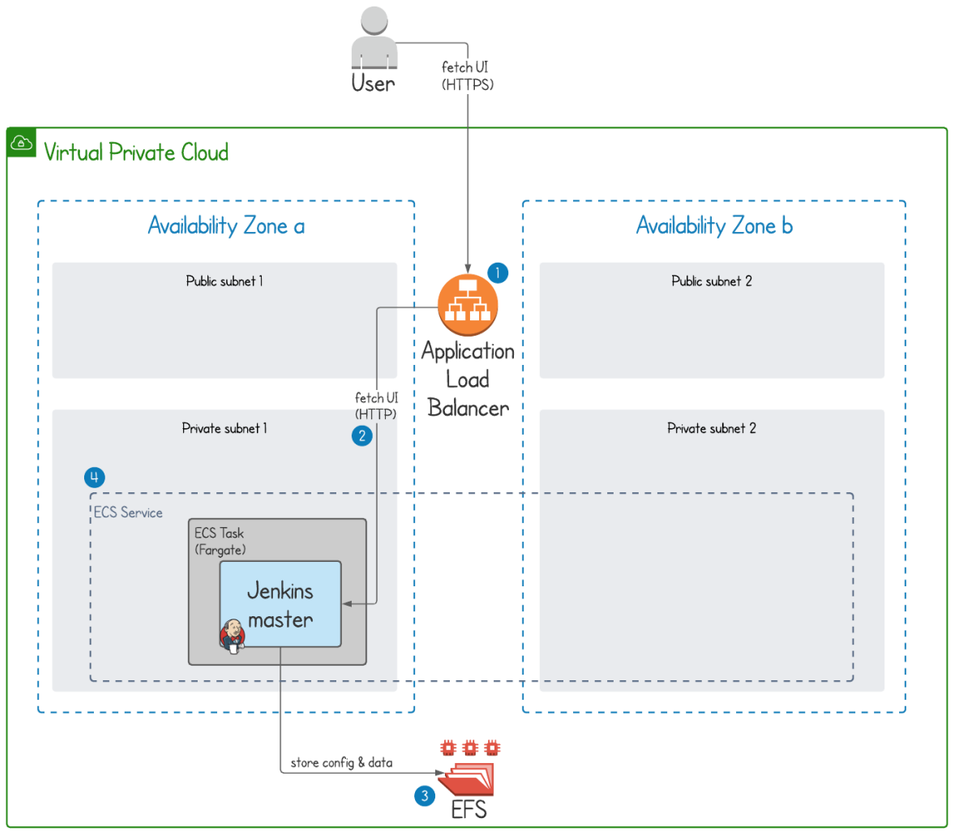
> **AWS & ECS lingo** 🤓
>
> AWS has a lot of magical acronyms and terms that can sometimes send your head in a spin. Let's get the relevant ones out in the open before continuing.
>
> - **ECS Task** – a unit of execution within ECS which equates to a Docker container running a single instance of an application
> - **ECS Service** – an orchestration layer, one for each type of application you want to deploy (e.g. Jenkins). It manages the ECS tasks for you, making sure the desired number are running, and handles security and networking.
> - **ECS Cluster** – a grouping of ECS tasks and services. A cluster can have a group of EC2 instances assigned to it on which it deploys tasks. This is called the EC2 launch type. Another launch type is Fargate, where AWS takes care of provisioning resources on which tasks run. Since you don't have to provision any EC2 instances yourself, it makes setup a lot simpler. We'll be using the Fargate launch type in this article.
> - **AWS Region** – a geographical area into which you deploy AWS resources, e.g. `eu-west-1` (Ireland), `eu-west-2` (London). We'll be using `eu-west-1` in this article.

- **AWS Availability Zone** – isolated datacentres within an AWS Region. Each region has multiple availability zones e.g. `eu-west-1` has `eu-west-1a`, `eu-west-1b`, and `eu-west-1c`.

3. A JENKINS SOLUTION IN ECS

OK, so we've given ECS the thumbs up, but let's think about what **specific features** we can use for our Jenkins deployment given the constraints of the Jenkins architecture described earlier. The following points are marked on the diagram below, 'cos I'm nice like that.

1. **Integration with Application Load Balancer (ALB)** – the only access point into Jenkins should be via an ALB which serves the Jenkins UI over HTTPS on port 443. An SSL certificate will be provided to the ALB for the domain we want Jenkins to be available on. Registration of ECS tasks into the ALB is handled automatically by ECS.
2. **Integration with Security Groups** – the Jenkins ECS service should be assigned a security group that only allows access on the Jenkins port (8080) from the ALB. We'll provide full outbound internet access to Jenkins in order that updates and plugins can be installed.
3. **Persistent storage** – AWS now offers tight integration between ECS tasks and the Elastic File System (EFS) service, meaning our Jenkins data will be safe if the container gets stopped for any reason.
4. **Failover** – because our Jenkins instance runs as a single master we can't run multiple instances of it, so it will be deployed into a single availability zone. Although problems with an AWS availability zone are rare, we can provide redundancy by creating an ECS service which spans multiple availability zones. This way, Jenkins will automatically recover if the availability zone it's running on fails.



EBS vs. EFS

Traditionally the **Elastic Block Store (EBS)** is the storage type to use when attaching a volume to an EC2 instance. This won't work for our use case, because:

- we're running in ECS Fargate, which doesn't support EBS
- an EBS volume can only exist in a single availability zone, making things difficult for our high availability requirement

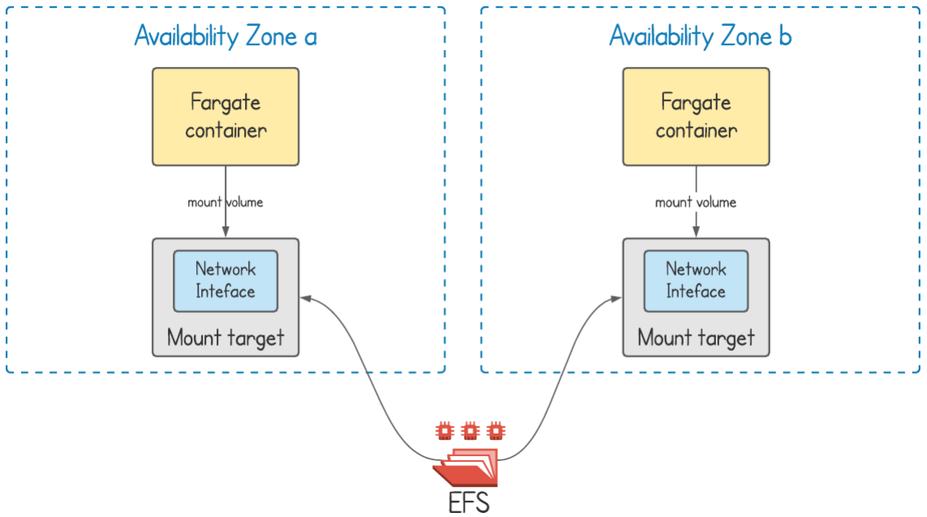
Fortunately though, the **Elastic File System (EFS)** can run in Fargate (as of April 2020) and its volumes can exist in multiple availability zones, which is why it's our choice for this article.

Attaching EFS to Fargate containers: 1,000 foot view

To keep this article on-point, I'll explain just enough information about attaching EFS to Fargate so that you can understand the main concepts. Sound fair?

EFS is an **Network File System** (NFS) type file system. It can be attached to one or many devices at the same time, each of which can read and write data. In the case of Fargate, you can attach an EFS file system to multiple ECS tasks.

The way this works with Fargate is using another resource that you have to create called a **mount target**. A mount target has its own network interface and therefore IP address, and it's via the mount target that an EFS resource is attached to a Fargate container.



You can see from the diagram above that each availability zone has its own mount target. EFS itself stores data across **multiple availability zones**. This means that if an availability zone fails, whatever Fargate containers are in the other availability zone keep running uninterrupted via the corresponding mount target.

As already mentioned we'll only have one Jenkins master running at once. But, these EFS features are very helpful to ensure Jenkins can come back automatically with the same data should an availability zone fail.

4. DEPLOYING JENKINS INTO ECS WITH CLOUDFORMATION

Let's get this show on the road by deploying all the AWS resources required to implement the solution above, using the AWS templating engine **CloudFormation**.

I recommend **first reading through the descriptions that follow** so you know what AWS resources you're deploying. But, if you're a super-keen eager beaver, jump right in by hitting **Launch Stack** below. This will create the CloudFormation stack in your own AWS account, resulting in a running Jenkins instance deployed in a new VPC and ECS cluster, available over the internet.

Full details of what to do when you click the Launch Stack button are given in the section Launching the CloudFormation stack in your AWS account.

Launch Stack

First though, let's run through the **two template files** that make up this infrastructure deployment.

Stack one: VPC and networking

This nested stack (default-vpc.yml) contains all the resources to create a standard VPC setup:

- **VPC** – a new network in the AWS cloud, where we'll be deploying all resources. Note that for us to attach an EFS volume to a Fargate container in this VPC, it must have **DNS hostnames** enabled.
- **public subnets x 2 (in different availability zones)** – here we'll deploy our ALB, so it's accessible to the internet
- **private subnets x 2 (in different availability zones)** – here we'll deploy our Jenkins service, so it's not directly accessible to the internet
- **an internet gateway** – attached to the public subnets, this is the network's route to the internet
- **NAT gateway x 2** – these allow traffic from any services deployed in the private subnets to reach the internet via the internet gateway

- **route tables, elastic IP, etc.** – see the CloudFormation template for full details of miscellaneous resources

To learn more about these AWS resources, see my guide **VPCs, subnets, and gateways – fundamentals for working with containers in AWS**.

> **Nested stacks**
>
> A nested stack is a reusable CloudFormation template. Because we're using a standard setup for the VPC/networking this has been extracted out into a separate nested stack, which can also be reused in future articles.

Stack two: ECS cluster, Jenkins ECS task, & ECS service

This main stack (jenkins-for-ecs.yml) references the nested stack created above, then it defines all the ECS resources required to get a Jenkins ECS service running, and hooks it into a load balancer.

ECS cluster

Our ECS cluster will be given the name default-cluster. Imaginative, I know!

```
ECSCluster:
  Type: AWS::ECS::Cluster
  Properties:
    ClusterName: default-cluster
```

ECS task definition

The Jenkins ECS task definition references the official Jenkins Docker image, and configures:

- `PortMappings` provides access to the container on port 8080
- `MountPoints` defines a mount point for a volume called jenkins-home inside the container at `/var/jenkins_home` (where Jenkins writes its data)
- the `LogConfiguration` sets up logging to CloudWatch using the `CloudwatchLogsGroup` resource, which keeps logs for 14 days
- the `Volumes` section contains a jenkins-home volume which uses the `EFSVolumeConfiguration` type to reference an EFS volume defined later on. Note that `TransitEncryption` is set to `ENABLED` so that Jenkins storage data is encrypted as it passes between the ECS task and EFS.

```
JenkinsTaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Family: !Sub jenkins-task
    Cpu: 512
    Memory: 1024
    NetworkMode: awsvpc
    TaskRoleArn: !Ref JenkinsRole
    ExecutionRoleArn: !Ref JenkinsExecutionRole
    RequiresCompatibilities:
      - FARGATE
      - EC2
    ContainerDefinitions:
      - Name: jenkins
        Image: jenkins/jenkins:lts
        PortMappings:
          - ContainerPort: 8080
        MountPoints:
          - SourceVolume: jenkins-home
            ContainerPath: /var/jenkins_home
        LogConfiguration:
          LogDriver: awslogs
          Options:
            awslogs-group: !Ref CloudwatchLogsGroup
            awslogs-region: !Ref AWS::Region
            awslogs-stream-prefix: jenkins
    Volumes:
      - Name: jenkins-home
        EFSVolumeConfiguration:
          FilesystemId: !Ref FileSystemResource
          TransitEncryption: ENABLED
          AuthorizationConfig:
            AccessPointId: !Ref AccessPointResource
            IAM: ENABLED
CloudwatchLogsGroup:
```

```
    Type: AWS::Logs::LogGroup
    Properties:
      LogGroupName: !Join ['-', [ECSLogGroup, !Ref 'AWS::StackName']]
      RetentionInDays: 14
```

ECS service

The Jenkins ECS service will be responsible for making sure the task (i.e. the container) is running, and it also manages networking:

- the `DesiredCount` of 1 means we'll get a single Jenkins instance
- the `LaunchType` of `FARGATE` means that AWS will be provisioning any underlying resources for us
- the `PlatformVersion` must be `1.4.0` otherwise the functionality to mount EFS volumes inside our Fargate container won't work
- the `DeploymentConfiguration` controls how ECS handles deployments when tasks need to be recreated. In our case we want at most one Jenkins instance to be running at once
- the provided `NetworkConfiguration` means the service can create ECS tasks in any of the given subnets. It also says the tasks should have network access restricted as described in the `JenkinsSecurityGroup`, which gives inbound access from the ALB on port 8080 (see template for details).
- the `LoadBalancers` section says the service should automatically register itself with the provided target group, defined in the next section

```
  JenkinsService:
    Type: AWS::ECS::Service
    DependsOn: LoadBalancerListener
    Properties:
      Cluster: !Ref ECSCluster
      TaskDefinition: !Ref JenkinsTaskDefinition
      DesiredCount: 1
      HealthCheckGracePeriodSeconds: 300
      LaunchType: FARGATE
      PlatformVersion: 1.4.0
      DeploymentConfiguration:
        MinimumHealthyPercent: 0
        MaximumPercent: 100
      NetworkConfiguration:
        AwsvpcConfiguration:
          AssignPublicIp: ENABLED
          Subnets:
            - !GetAtt VPCStack.Outputs.PrivateSubnet1
            - !GetAtt VPCStack.Outputs.PrivateSubnet2
          SecurityGroups:
            - !GetAtt JenkinsSecurityGroup.GroupId
      LoadBalancers:
        - ContainerName: jenkins
          ContainerPort: 8080
          TargetGroupArn: !Ref JenkinsTargetGroup
```

Load balancer and related resources

To expose Jenkins to the internet over SSL we'll create a load balancer and a load balancer listener to receive incoming HTTPS traffic.

- the LoadBalancer spans two public subnets
- it's assigned a `LoadBalancerSecurityGroup` which allows inbound traffic from the internet on the SSL port 443, and outbound traffic to our Jenkins instance only
- the `LoadBalancerListener`:
  - listens for HTTPS traffic on port 443
  - is assigned a certificate id which must be passed into the CloudFormation template as a parameter (see *Certificate & DNS setup* below)
  - by default forwards traffic to the Jenkins target group
- the `JenkinsTargetGroup` is where the ECS service will register the Jenkins task IP address
  - the `HealthCheckpath` is `/login` which returns a 200
  - the `Protocol` at this point is `HTTP` since by this point traffic has been decrypted by the ALB
  - `deregistration_delay` is how long the target group will wait for requests to drain from a target when it's being deregistered. Reducing this from the default of 5 minutes to 10 seconds means that changes to the Jenkins task will happen quicker.

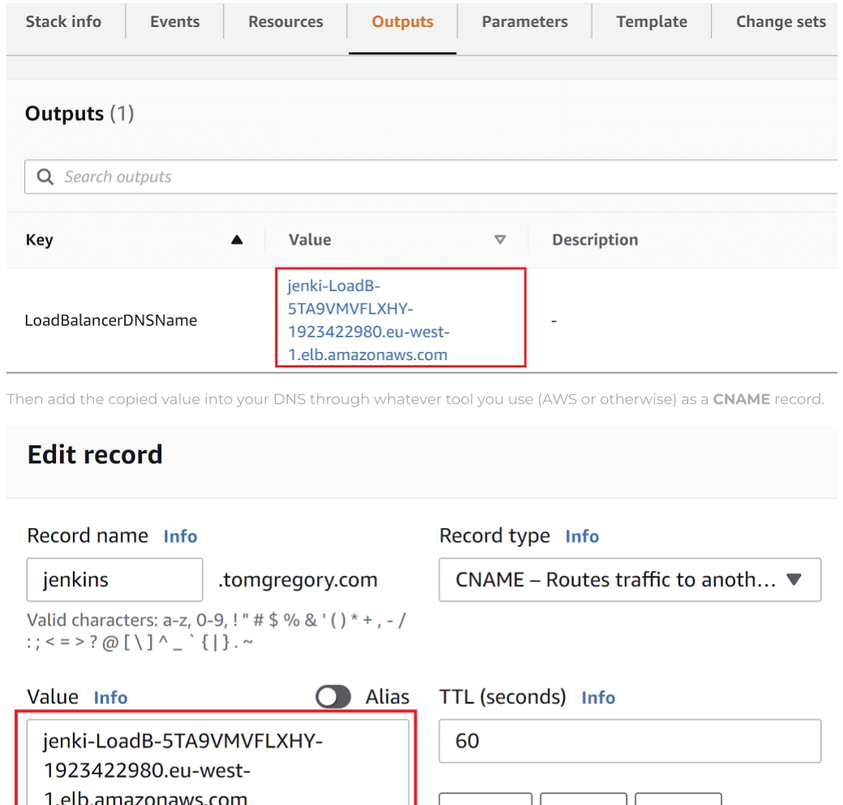
```
LoadBalancer:
  Type: AWS::ElasticLoadBalancingV2::LoadBalancer
  Properties:
    Subnets:
      - !GetAtt VPCStack.Outputs.PublicSubnet1
      - !GetAtt VPCStack.Outputs.PublicSubnet2
    SecurityGroups:
      - !Ref LoadBalancerSecurityGroup
LoadBalancerSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupName: LoadBalancerSecurityGroup
    GroupDescription: Security group for load balancer
    VpcId: !GetAtt VPCStack.Outputs.VPC
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 443
        ToPort: 443
        CidrIp: 0.0.0.0/0
    SecurityGroupEgress:
      - IpProtocol: tcp
        FromPort: 8080
        ToPort: 8080
        DestinationSecurityGroupId: !Ref JenkinsSecurityGroup
LoadBalancerListener:
  Type: AWS::ElasticLoadBalancingV2::Listener
  Properties:
    Certificates:
      - CertificateArn: !Ref CertificateArn
    DefaultActions:
      - Type: forward
        ForwardConfig:
          TargetGroups:
            - TargetGroupArn: !Ref JenkinsTargetGroup
    LoadBalancerArn: !Ref LoadBalancer
    Port: 443
    Protocol: HTTPS
JenkinsTargetGroup:
  Type: AWS::ElasticLoadBalancingV2::TargetGroup
  Properties:
    HealthCheckPath: /login
    Name: JenkinsTargetGroup
    Port: 8080
    Protocol: HTTP
    TargetType: ip
    VpcId: !GetAtt VPCStack.Outputs.VPC
    TargetGroupAttributes:
      - Key: deregistration_delay.timeout_seconds
        Value: 10
```

Certificate & DNS setup

You can use an existing certificate or create one through **Services > Certificate Manager**. Create the certificate for whatever domain you want Jenkins to be available on. In my case I wanted to serve Jenkins on `jenkins.tomgregory.com` so created a wildcard certificate for `*.tomgregory.com`.

Once your domain has been validated and the certificate has been created, the certificate ARN should be passed as a parameter when launching the CloudFormation stack (see *Launching the CloudFormation stack in your AWS account* below).

After the stack has been created, on the stack details page select the **Outputs** tab and copy the value from **LoadBalancerDNSName**. This is the DNS name of the load balancer created by CloudFormation.



EFS File System

- the `EFSSecurityGroup` provides access from the Jenkins security group on port 2049, the default EFS port
- the `FileSystemResource` is the volume itself, which we provide a name of `jenkins-home`
- `Encrypted` is set to true to enable encryption at rest
- two mount targets `MountTargetResource1` and `MountTargetResource2` provide access from ECS tasks to the file system. We create a mount target in each of our private subnets, so depending on which availability zone our Jenkins ECS task gets placed, it will always have access to the file system.

```
EFSSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    VpcId: !GetAtt VPCStack.Outputs.VPC
    GroupDescription: Enable EFS access via port 2049
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 2049
        ToPort: 2049
        SourceSecurityGroupId: !Ref JenkinsSecurityGroup
FileSystemResource:
  Type: AWS::EFS::FileSystem
  Properties:
    Encrypted: true
    FileSystemTags:
      - Key: Name
        Value: jenkins-home
MountTargetResource1:
  Type: AWS::EFS::MountTarget
  Properties:
    FileSystemId: !Ref FileSystemResource
    SubnetId: !GetAtt VPCStack.Outputs.PrivateSubnet1
    SecurityGroups:
```

```
      - !GetAtt EFSSecurityGroup.GroupId
  MountTargetResource2:
    Type: AWS::EFS::MountTarget
    Properties:
      FileSystemId: !Ref FileSystemResource
      SubnetId: !GetAtt VPCStack.Outputs.PrivateSubnet2
      SecurityGroups:
        - !GetAtt EFSSecurityGroup.GroupId
  AccessPointResource:
    Type: AWS::EFS::AccessPoint
    Properties:
      FileSystemId: !Ref FileSystemResource
      PosixUser:
        Uid: '1000'
        Gid: '1000'
      RootDirectory:
        CreationInfo:
          OwnerGid: '1000'
          OwnerUid: '1000'
          Permissions: '755'
        Path: '/jenkins-home'
```

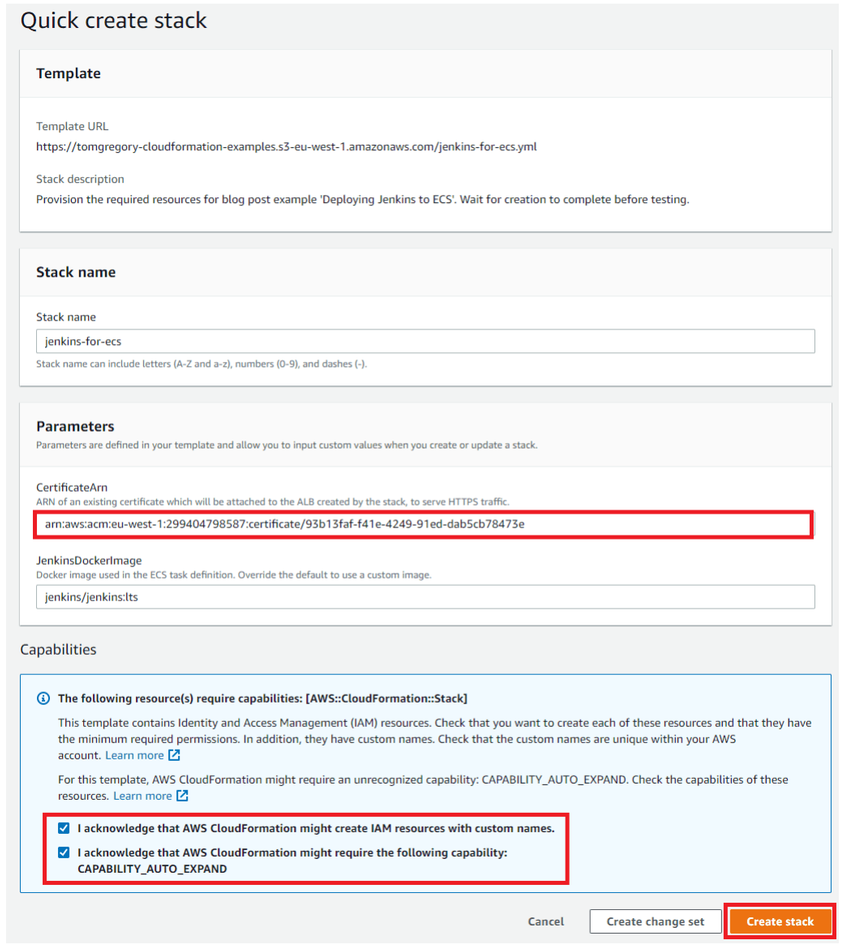
5. LAUNCHING THE CLOUDFORMATION STACK IN YOUR AWS ACCOUNT

Now you know what resources are created in the CloudFormation templates you can go ahead and deploy the example using the Launch Stack button below:

When you click this button you'll be taken to the **Quick create stack** page in your own AWS account.

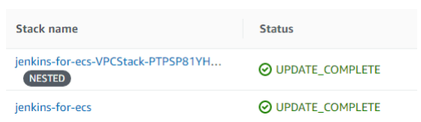
- provide a **CertificateArn**. This is the ARN of the certificate you want to attach to your ALB for HTTPS access (see the Certificates & DNS setup section for more details).
- accept that this stack may create IAM resources and needs the `CAPABILITY_AUTO_EXPAND` capability

Click **Create stack**.



AWS will now go off and do its business, the infrastructure type business that is.

Go to **Services > CloudFormation** and you'll see your CloudFormation stacks in the process of being created. After about 5 minutes all the stacks should be in the `UPDATE_COMPLETE` state.



Prod ready? Yes, but...

The CloudFormation I've supplied here deploys **an example** of a production ready Jenkins service into a new VPC. In applying this to your own production infrastructure, I suggest:

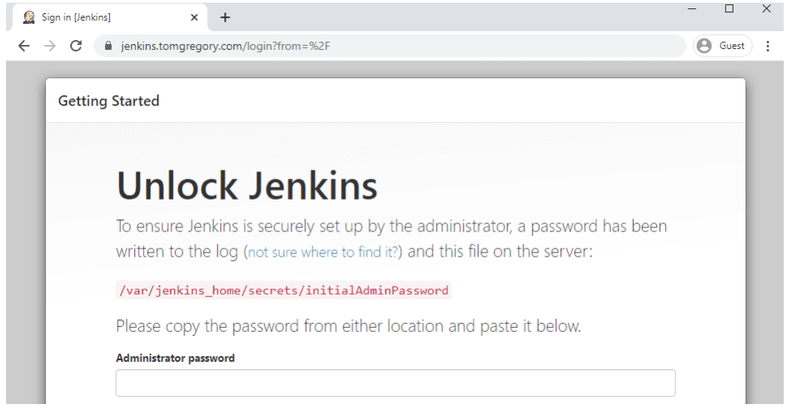
1. making a copy of any CloudFormation templates provided here and hosting them in your own S3 bucket
2. adjusting as necessary to integrate with your own VPC and subnets
3. setting up log retention and automated EFS backups based on your own requirements
4. limiting access to your Jenkins instance by IP if you can. This can be achieved by updating the `LoadBalancerSecurityGroup` ingress rule.

6. GETTING STARTED WITH THE JENKINS INSTANCE

If you've applied the CloudFormation stack described above you'll now have a shiny new Jenkins instance running. Be patient as even when the stack has applied Jenkins still takes a couple of minutes to start up. ⏰

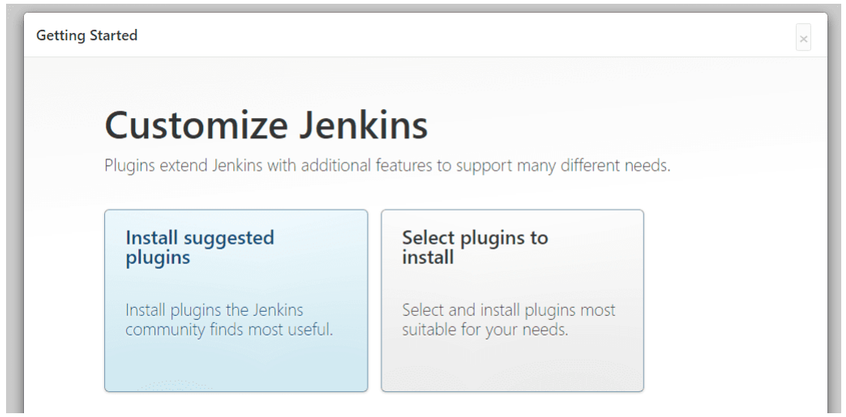
You can access Jenkins using the ALB DNS name described in the Certificate & DNS setup section above, although you'll have to accept the *"Your connection is not private"* warning about an insecure certificate. If you've setup your certificate and DNS correctly

though, you'll be able to access Jenkins with a valid certificate for the domain, which will look like this:



Grab the admin password by navigating to the ECS Task and clicking on the **Logs** tab. The password will be printed in the logs the first time Jenkins starts up. Look for the log line *"Please use the following password to proceed to installation"*.

Copy the password, paste it into Jenkins, and you'll be off and away with the Jenkins setup wizard:



Once you've followed this through, you'll have a Jenkins instance ready to start running some jobs!

7. DISASTER RECOVERY SCENARIOS

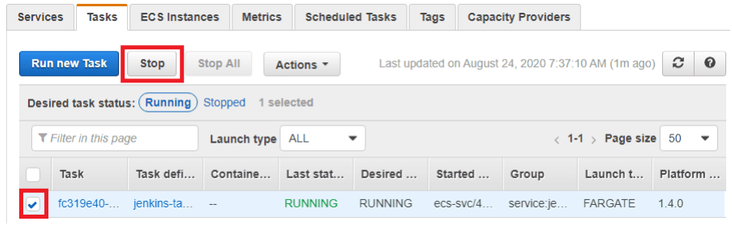
Since the intention of this article is to create a **production-ready** Jenkins deployment, let's put our money where our mouth is and test some disaster recovery scenarios.

1. ECS task failure

Scenario: the Jenkins ECS Task gets stopped for some reason

Requirement: the ECS task should restart automatically restoring service quickly

Test: go to the list of tasks in the cluster, select the Jenkins task, then click **Stop**

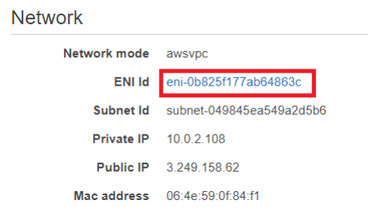


Observation: ECS restarts the task and Jenkins is available again in 2m00s. ✅

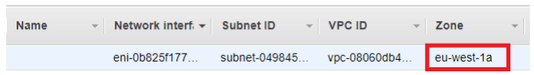
2. Availability zone failure

This is a real "squeaky bum time" scenario where an entire AWS availability zone datacentre goes down.

You can identify which availability zone your ECS task is running in by going to the task details page and clicking on the **ENI Id** under **Network**:



This opens up details of the **Elastic Network Interface** (ENI) which our ECS task uses to connect to the network and internet. Under **Zone** it tells us which availability zone this ENI and associated ECS task are in.



In my case I know then that I need to simulate a failure of **eu-west-1a**.

Sadly AWS won't be very helpful in bringing down a whole availability zone for our testing. 😢 But, the next best thing we can do is to modify our Jenkins ECS service to force it to deploy into a different subnet and therefore a different availability zone.

In the main CloudFormation template *jenkins-for-ecs.yml* it passes a list of subnet ids through in the `NetworkConfiguration` section of the `JenkinsService` resource:

```
NetworkConfiguration:
  AwsvpcConfiguration:
    AssignPublicIp: ENABLED
    Subnets:
      - !GetAtt VPCStack.Outputs.PrivateSubnet1
      - !GetAtt VPCStack.Outputs.PrivateSubnet2
    SecurityGroups:
      - !GetAtt JenkinsSecurityGroup.GroupId
```

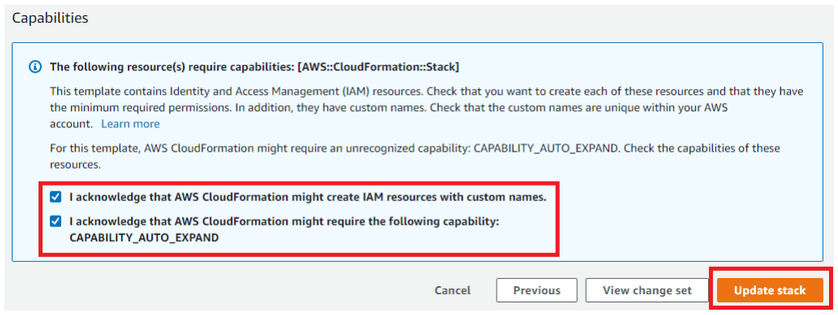
All we need to do is tweak this list so that it only contains *PrivateSubnet2*, which lives in the second availability zone in your region (in my case **eu-west-1b**).

```
NetworkConfiguration:
  AwsvpcConfiguration:
    AssignPublicIp: ENABLED
    Subnets:
      - !GetAtt VPCStack.Outputs.PrivateSubnet2
    SecurityGroups:
      - !GetAtt JenkinsSecurityGroup.GroupId
```

Once this change is applied the ECS service creates the ECS task in the remaining subnet, and the Jenkins instance is available again in 2m28s. ✅

Making these changes yourself

Download the original jenkins-for-ecs.yml template and make the above code change. To apply it, go to **Services > CloudFormation**, then select the stack and click **Update**. You can then select **Replace current template** and upload the file. Keep clicking **Next** then on the final screen click to say you accept the additional capabilities, then click **Update stack**.



8. NEXT STEPS

Even though now you have created a production-ready Jenkins instance which is **secure** and **highly available**, there's still some way to go before you have a full **continuous integration** solution in place. Check out the next article in this series *Running Jenkins jobs in AWS ECS with slave agents* where we'll learn how to horizontally scale Jenkins workloads by running jobs on slave agents.

As a final point, if you followed the example in this tutorial don't forget to delete the CloudFormation stack once you're finished with it, to avoid incurring unnecessary charges.

9. RESOURCES

CloudFormation

The CloudFormation stack can be applied directly to your AWS account by clicking below:

There were two templates used in this article:

1. jenkins-for-ecs.yml (main template)
2. default-vpc.yml (nested)

AWS

- for more detailed info on how EFS connects with Fargate, check out AWS's in-depth *Developers guide to using Amazon EFS with Amazon ECS and AWS Fargate*
- to learn more about VPC setup and related resources, read the AWS article *What is Amazon VPC?*

Jenkins

The Docker image used in this article was **jenkins/jenkins**, available on Docker Hub

Video

Check out the accompanying video over on my YouTube channel