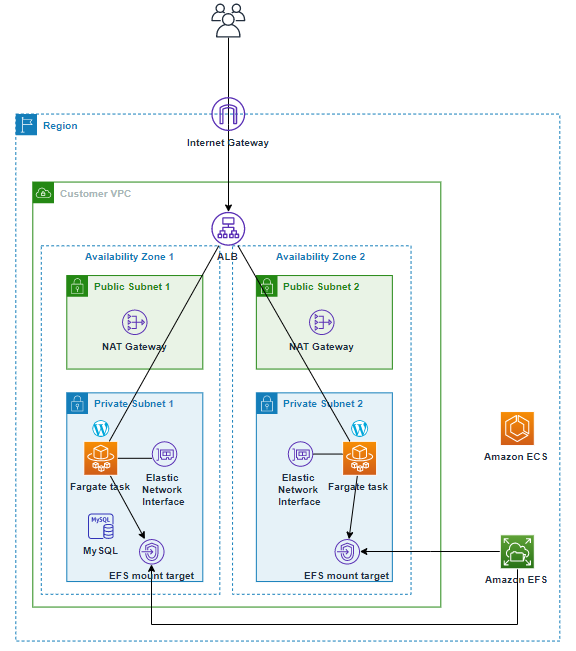
**Вводные и принятие решения о методе реализации.**

Так как инстанс Wordpress является ярким примером stateless деплоймента, где вся информация храниться только в базе данных и распределенной файловой системе, было принято решение реализовать задачу по построению отказоустойчивого кластера Wordpress на основе Amazon Elastic Container Service (ECS). ESC очень хорошо подходит для реализации stateless приложений.

В качестве общей базы данных используется RDS db.t3.micro mysql, а для распределенной файловой системы – EFS с монтированием файловой системы в контейнеры.

Для развертывания инфраструктуры используется aws cli и Aws Cloudformation.

Примерная схема планируемого решения.



Доступ в WP, работающий в контейнерах в ECS (в разные зонах доступности) в приватной сети происходит с помощь aws ELB , который ссылается на Target group WP ecs кластера. Приватные сети в которых работает WP изолированы и достану во вне идет через отдельные для каждой подсети NAT Gateway.

**Порядок установки и настройки кластера Wordpress.**

1. Импортируем общие переменный окружения для инсталляции.

|  |
| --- |
| export WP\_AWS\_REGION=eu-central-1 && \  export WP\_ACCOUNT\_ID=$(aws sts get-caller-identity --query 'Account' --output text) && \  export WP\_ECS\_CLUSTER\_NAME=Ecs-WordPress && \  export WP\_CFN\_STACK\_NAME=WordPress-Aws-Task |

1. Создаем нашу aws cloudformation инфраструктуру на основе ***wordpress-ecs.yaml*.**

|  |
| --- |
| aws cloudformation create-stack \  --stack-name $WP\_CFN\_STACK\_NAME \  --region $WP\_AWS\_REGION \  --template-body file://wordpress-ecs.yaml |

Следить за готовностью создания ресурсов можно с помощью команды.

|  |
| --- |
| aws cloudformation wait stack-create-complete \  --stack-name $(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query 'Stacks[0].StackId' --output text) \  --region $WP\_AWS\_REGION |

На этом этапе будут создана VPC со двумя зонами доступности, в каждой по две сети, приватная и публичная и все необходимое для их работы (NAT GW, EIP, GW, ELB) . Инстанс EFS и инстанс RDS.

Значения полученных настроек, вроде "VPCId of VPC" будут помещены в указанные Outputs.

1. После создания нашей инфраструктуры выполняем команды для импортирования переменных с полученными настройками из cloudformation Outputs в переменные окружения.

|  |
| --- |
| export WP\_EFS\_FS\_ID=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='EFSFSId'].OutputValue" \  --output text)  export WP\_EFS\_AP=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='EFSAccessPoint'].OutputValue" \  --output text)    export WP\_RDS\_ENDPOINT=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='RDSEndpointAddress'].OutputValue" \  --output text)    export WP\_VPC\_ID=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='VPCId'].OutputValue" \  --output text)    export WP\_PUBLIC\_SUBNET0=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='PublicSubnet0'].OutputValue" \  --output text)    export WP\_PUBLIC\_SUBNET1=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='PublicSubnet1'].OutputValue" \  --output text)    export WP\_PRIVATE\_SUBNET0=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='PrivateSubnet0'].OutputValue" \  --output text)  export WP\_PRIVATE\_SUBNET1=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='PrivateSubnet1'].OutputValue" \  --output text)    export WP\_ALB\_SG\_ID=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='ALBSecurityGroup'].OutputValue" \  --output text)    export WP\_TG\_ARN=$(aws cloudformation describe-stacks \  --region $WP\_AWS\_REGION \  --stack-name $WP\_CFN\_STACK\_NAME \  --query "Stacks[0].Outputs[?OutputKey=='WordPressTargetGroup'].OutputValue" \  --output text) |

1. Создаем файл описания таска для создания на основе него инcтансов WP в ecs. Используются ранее экспортированные переменные окружения.

|  |
| --- |
| cat > wp-task-definition.json << EOF  { "networkMode": "awsvpc",  "containerDefinitions": [  {    "portMappings": [  {  "containerPort": 8080,  "protocol": "tcp"  }  ],  "essential": true,  "mountPoints": [  {  "containerPath": "/bitnami/wordpress",  "sourceVolume": "wordpress"  }  ],  "name": "wordpress",  "image": "bitnami/wordpress",  "environment": [  {  "name": "MARIADB\_HOST",  "value": "${WP\_RDS\_ENDPOINT}"  },  {  "name": "WORDPRESS\_DATABASE\_USER",  "value": "admin"  },  {  "name": "WORDPRESS\_DATABASE\_PASSWORD",  "value": "supersecretpassword"  },  {  "name": "WORDPRESS\_DATABASE\_NAME",  "value": "wordpress"  },  {  "name": "PHP\_MEMORY\_LIMIT",  "value": "512M"  },  {  "name": "enabled",  "value": "false"  },  {  "name": "ALLOW\_EMPTY\_PASSWORD",  "value": "yes"  }  ]  }  ],  "requiresCompatibilities": [  "FARGATE"  ],  "cpu": "1024",  "memory": "3072",  "volumes": [  {  "name": "wordpress",  "efsVolumeConfiguration": {  "fileSystemId": "${WP\_EFS\_FS\_ID}",  "transitEncryption": "ENABLED",  "authorizationConfig": {  "accessPointId": "${WP\_EFS\_AP}",  "iam": "DISABLED"  }  }  }  ],  "family": "wp-task"  }  EOF |

Здесь указываются ранее созданные база данных в RDS (через переменные окружения) и общая файловая система EFS , которая монтируется в "/bitnami/wordpress".

1. Регистрируем описанный таск WP.

|  |
| --- |
| WP\_TASK\_DEFINITION\_ARN=$(aws ecs register-task-definition \  --cli-input-json file://wp-task-definition.json \  --region $WP\_AWS\_REGION \  --query taskDefinition.taskDefinitionArn --output text) |

1. Создаем кластер ESC и группы безопасности для трафика на 8080 порту приходящего из балансера.

|  |
| --- |
| aws ecs create-cluster \  --cluster-name $WP\_ECS\_CLUSTER\_NAME \  --region $WP\_AWS\_REGION |

|  |
| --- |
| WP\_SVC\_SG\_ID=$(aws ec2 create-security-group \  --description Svc-WordPress-ECS \  --group-name Svc-WordPress-ECS \  --vpc-id $WP\_VPC\_ID --region $WP\_AWS\_REGION \  --query 'GroupId' --output text)  ##Accept traffic on port 8080  aws ec2 authorize-security-group-ingress \  --group-id $WP\_SVC\_SG\_ID --protocol tcp \  --port 8080 --source-group $WP\_ALB\_SG\_ID \  --region $WP\_AWS\_REGION |

1. Создаем инстанс WP в ECS.

|  |
| --- |
| aws ecs create-service \  --cluster $WP\_ECS\_CLUSTER\_NAME \  --service-name wp-efs-rw-service \  --task-definition "${WP\_TASK\_DEFINITION\_ARN}" \  --load-balancers targetGroupArn="${WP\_TG\_ARN}",containerName=wordpress,containerPort=8080 \  --desired-count 2 \  --platform-version 1.4.0 \  --launch-type FARGATE \  --deployment-configuration maximumPercent=100,minimumHealthyPercent=0 \  --network-configuration "awsvpcConfiguration={subnets=["$WP\_PRIVATE\_SUBNET0,$WP\_PRIVATE\_SUBNET1"],securityGroups=["$WP\_SVC\_SG\_ID"],assignPublicIp=DISABLED}"\  --region $WP\_AWS\_REGION |

1. Дожидаемся создания тасков и проверяем доступность.

|  |
| --- |
| echo "http://$(aws elbv2 describe-load-balancers \  --names wp-load-balancer --region $WP\_AWS\_REGION \  --query 'LoadBalancers[].DNSName' --output text)/wp-admin/" |

Получаем ссылку на админку WP. Пользователь “user” пароль “bitnami”. Не забываем поменять админский пароль.

1. Проверка отказоустойчивости.

После небольших настроек, можно осткалировать наш кластер в 0.

|  |
| --- |
| aws ecs update-service \  --cluster $WP\_ECS\_CLUSTER\_NAME \  --region $WP\_AWS\_REGION \  --service wp-efs-rw-service \  --task-definition "$WP\_TASK\_DEFINITION\_ARN" \  --desired-count 0 |

Дождаться остановки тасков и вернуть --desired-count обратно в значение 2.

После этого мы можем видеть, что все сделанные нами изменения остались на месте.

1. Как вариант, еще можно добавить возможность автоматического горизонтального скалирование инстанса ecs в зависимости от нагрузки.

Регистрируем таргет для автоскелинга, и желаемое минимально и максимальное количество инстансов.

|  |
| --- |
| aws application-autoscaling \  register-scalable-target \  --region $WP\_AWS\_REGION \  --service-namespace ecs \  --resource-id service/${WP\_ECS\_CLUSTER\_NAME}/wp-efs-rw-service \  --scalable-dimension ecs:service:DesiredCount \  --min-capacity 2 \  --max-capacity 4 |

Создаем политику скалирования в json файле.

|  |
| --- |
| cat > scaling.config.json << EOF  {  "TargetValue": 75.0,  "PredefinedMetricSpecification": {  "PredefinedMetricType": "ECSServiceAverageCPUUtilization"  },  "ScaleOutCooldown": 60,  "ScaleInCooldown": 60  }  EOF --max-capacity 4 |

Применяем полученную политику.

|  |
| --- |
| aws application-autoscaling put-scaling-policy \  --service-namespace ecs \  --scalable-dimension ecs:service:DesiredCount \  --resource-id service/${WP\_ECS\_CLUSTER\_NAME}/wp-efs-rw-service \  --policy-name cpu75-target-tracking-scaling-policy \  --policy-type TargetTrackingScaling \  --region $WP\_AWS\_REGION \  --target-tracking-scaling-policy-configuration file://scaling.config.json |

1. После всех проверок не забываем удалить все созданные нами инстансы.

|  |
| --- |
| aws application-autoscaling delete-scaling-policy --policy-name cpu75-target-tracking-scaling-policy --service-namespace ecs --resource-id service/${WP\_ECS\_CLUSTER\_NAME}/wp-efs-rw-service --scalable-dimension ecs:service:DesiredCount --region $WP\_AWS\_REGION && \  aws application-autoscaling deregister-scalable-target --service-namespace ecs --resource-id service/${WP\_ECS\_CLUSTER\_NAME}/wp-efs-rw-service --scalable-dimension ecs:service:DesiredCount --region $WP\_AWS\_REGION && \  aws ecs delete-service --service wp-efs-rw-service --cluster $WP\_ECS\_CLUSTER\_NAME --region $WP\_AWS\_REGION --force && \  aws ec2 revoke-security-group-ingress --group-id $WP\_SVC\_SG\_ID --region $WP\_AWS\_REGION --protocol tcp --port 8080 --source-group $WP\_ALB\_SG\_ID && \  aws ec2 delete-security-group --group-id $WP\_SVC\_SG\_ID --region $WP\_AWS\_REGION && \  aws ecs delete-cluster --cluster $WP\_ECS\_CLUSTER\_NAME --region $WP\_AWS\_REGION && \  aws cloudformation delete-stack --stack-name $WP\_CFN\_STACK\_NAME --region $WP\_AWS\_REGION |

**Скриншоты работающего приложения и настроек AWS можно увидеть в папке Images.**