Published in Level Up Coding

BoNeZ Follow
Aug 18, 2022 · 8 min read · ✦ · Listen

Django app deployed using GitLab pipeline on AWS ECS

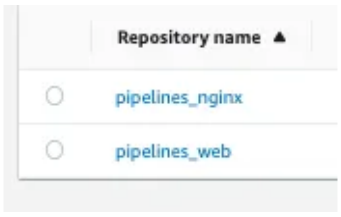
As a newbie setting up pipelines in GitLab can be cumbersome. I shall walk you through a step by step instruction of a basic flow to deploy to an AWS ECS container service when a merge request occurs to a specific branch. I will be using a Django application for this example.

PreRequisite:

- AWS
- Django
- Docker
- Web service

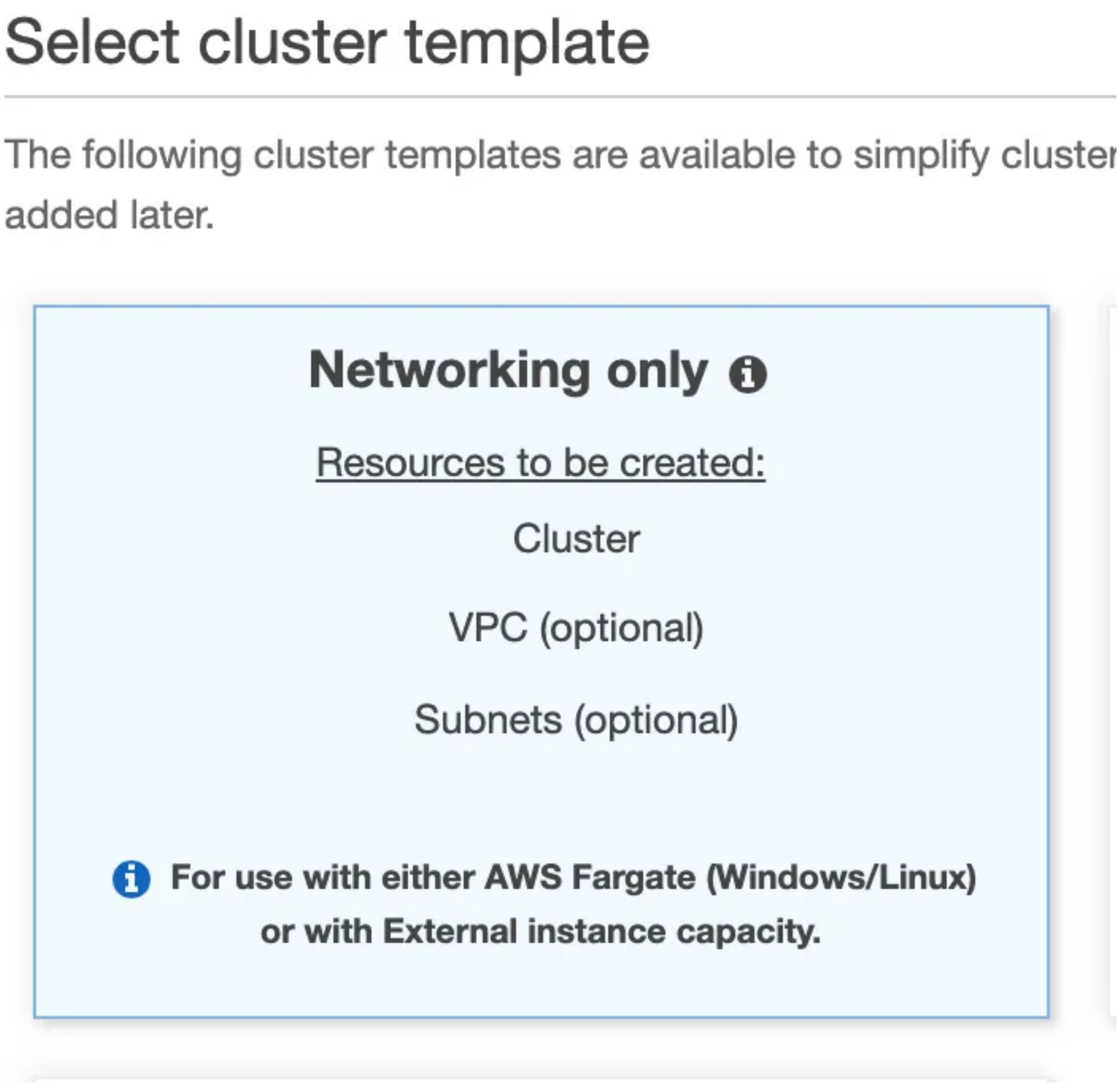
Our first goal is to try get the application working locally using Docker with a web-service, preferably on a different port than the usual 80. I will use Nginx in this example.

Check you have ECS repositories in https://eu-west-2.console.aws.amazon.com/ecr/repositories?region=eu-west-2. You will need 2 one for application and one for web service. This is where ECS will use the docker images to load from defined further below



Repository name ▲
pipelines_nginx
pipelines_web

Now to create a new test cluster, service and tasks.I will use all defaults


Select cluster template

The following cluster templates are available to simplify cluster
added later.

Networking only

Resources to be created:

Cluster

VPC (optional)

Subnets (optional)

For use with either AWS Fargate (Windows/Linux) or with External instance capacity.

Create a new Task, you can use the wizard or json. Replace *** as appropriate.

The values in the json key object `environment` are the ones required for your application. I may have some extra, which are not needed for a basic application.

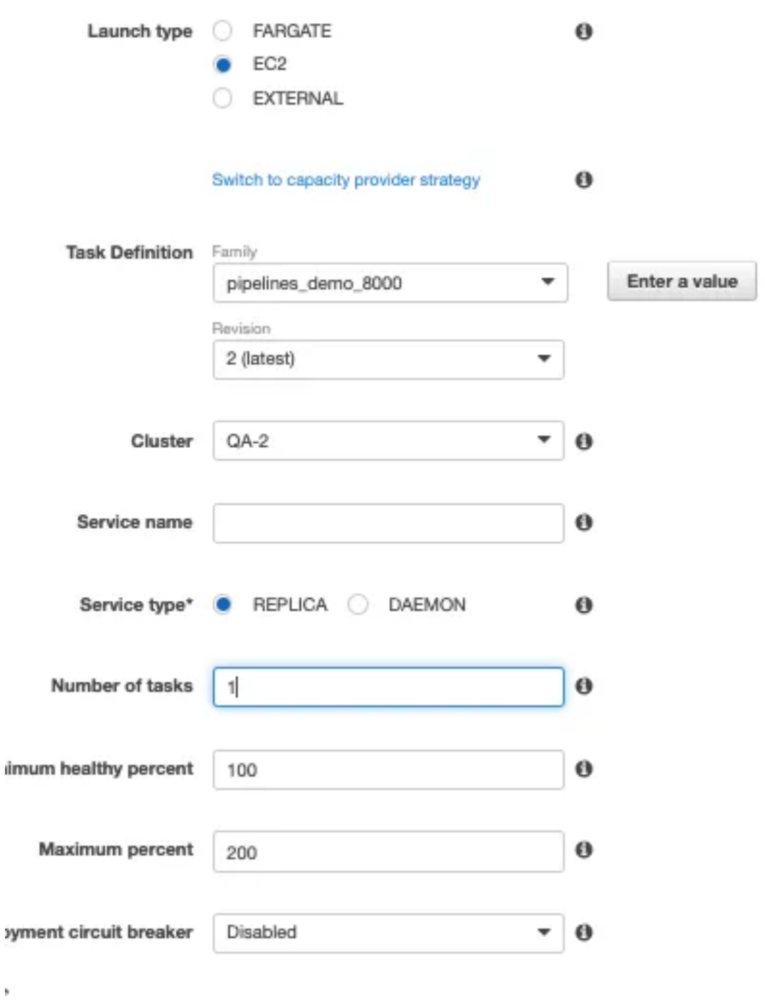
I have enabled `logConfiguration` as this is useful to debug you environment in AWS Cloudwatch.

For `portMappings` you want to set high port which will be used to access the application from your browser. Within the container it self it will be port 80 as normal.

There are are 2 `containerDefinitions` one for web service and one from application. they will use the repositories created earlier, specified under the key `image`

```
{
  "ipcMode": null,
  "executionRoleArn": null,
  "containerDefinitions": [
    {
      "dnsSearchDomains": null,
      "environmentFiles": null,
      "logConfiguration": {
        "logDriver": "awslogs",
        "secretOptions": null,
        "options": {
          "awslogs-group": "/ecs/pipelines_4000",
          "awslogs-region": "eu-west-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "entryPoint": null,
      "portMappings": [
        {
          "hostPort": 4000,
          "protocol": "tcp",
          "containerPort": 4000
        }
      ],
      "command": null,
      "linuxParameters": null,
      "cpu": 0,
      "environment": [
        {
          "name": "ALLOWED_HOSTS",
          "value": "*"
        },
        {
          "name": "AWS_ACCESS_KEY_ID",
          "value": "***"
        },
        {
          "name": "AWS_S3_REGION_NAME",
          "value": "eu-west-2"
        },
        {
          "name": "AWS_SECRET_ACCESS_KEY",
          "value": "***"
        },
        {
          "name": "AWS_STORAGE_BUCKET_NAME",
          "value": "***"
        },
        {
          "name": "BASE_URL",
          "value": "***"
        },
        {
          "name": "DEBUG",
          "value": "1"
        },
        {
          "name": "DEFAULT_FROM_EMAIL",
          "value": "***"
        },
        {
          "name": "DEFAULT_SUPERUSER_EMAIL",
          "value": "***"
        },
        {
          "name": "DEFAULT_SUPERUSER_PASSWORD",
          "value": "***"
        },
        {
          "name": "DEFAULT_SUPERUSER_USERNAME",
          "value": "reach_super"
        },
        {
          "name": "EMAIL_BACKEND",
          "value": "django.core.mail.backends.smtp.EmailBackend"
        },
        {
          "name": "EMAIL_HOST",
          "value": "***"
        },
        {
          "name": "EMAIL_HOST_PASSWORD",
          "value": "***"
        },
        {
          "name": "EMAIL_HOST_USER",
          "value": "***"
        },
        {
          "name": "EMAIL_PORT",
          "value": "***"
        },
        {
          "name": "EMAIL_USE_SSL",
          "value": "0"
        },
        {
          "name": "EMAIL_USE_TLS",
          "value": "1"
        },
        {
          "name": "ENVIRONMENT",
          "value": "***"
        },
        {
          "name": "RDS_DB_NAME",
          "value": "***"
        },
        {
          "name": "RDS_HOSTNAME",
          "value": "***"
        },
        {
          "name": "RDS_PASSWORD",
          "value": "***"
        },
        {
          "name": "RDS_PORT",
          "value": "***"
        },
        {
          "name": "RDS_USERNAME",
          "value": "***"
        },
        {
          "name": "SECRET_KEY",
          "value": "***"
        },
        {
          "name": "SESSION_COOKIE_DOMAIN",
          "value": "***"
        }
      ],
      "resourceRequirements": null,
      "ulimits": null,
      "dnsServers": null,
      "mountPoints": [],
      "workingDirectory": null,
      "secrets": null,
      "dockerSecurityOptions": null,
      "memory": 350,
      "memoryReservation": null,
      "volumesFrom": [],
      "stopTimeout": null,
      "image": "***.dkr.ecr.eu-west-2.amazonaws.com/pipelines-
django",
      "startTimeout": null,
      "firelensConfiguration": null,
      "dependsOn": null,
      "disableNetworking": null,
      "interactive": null,
      "healthCheck": null,
      "essential": true,
      "links": null,
      "hostname": null,
      "extraHosts": null,
      "pseudoTerminal": null,
      "user": null,
      "readonlyRootFilesystem": null,
      "dockerLabels": null,
      "systemControls": null,
      "privileged": null,
      "name": "pipelines_django"
    },
    {
      "dnsSearchDomains": null,
      "environmentFiles": null,
      "logConfiguration": {
        "logDriver": "awslogs",
        "secretOptions": null,
        "options": {
          "awslogs-group": "/ecs/pipelines_4000",
          "awslogs-region": "eu-west-2",
          "awslogs-stream-prefix": "ecs"
        }
      },
      "entryPoint": null,
      "portMappings": [
        {
          "hostPort": 4001,
          "protocol": "tcp",
          "containerPort": 80
        }
      ],
      "command": null,
      "linuxParameters": null,
      "cpu": 0,
      "environment": [],
      "resourceRequirements": null,
      "ulimits": null,
      "dnsServers": null,
      "mountPoints": [],
      "workingDirectory": null,
      "secrets": null,
      "dockerSecurityOptions": null,
      "memory": 128,
      "memoryReservation": null,
      "volumesFrom": [],
      "stopTimeout": null,
      "image": "***.dkr.ecr.eu-west-2.amazonaws.com/pipelines-nginx",
      "startTimeout": null,
      "firelensConfiguration": null,
      "dependsOn": null,
      "disableNetworking": null,
      "interactive": null,
      "healthCheck": null,
      "essential": true,
      "links": [
        "pipelines_django"
      ],
      "hostname": null,
      "extraHosts": null,
      "pseudoTerminal": null,
      "user": null,
      "readonlyRootFilesystem": null,
      "dockerLabels": null,
      "systemControls": null,
      "privileged": null,
      "name": "pipelines_nginx"
    }
  ],
  "placementConstraints": [],
  "memory": null,
  "taskRoleArn": null,
  "compatibilities": [
    "EXTERNAL",
    "EC2"
  ],
  "taskDefinitionArn": "arn:aws:ecs:eu-west-2:***:task-
definition/pipelines_4000:8",
  "family": "pipelines_4000",
  "requiresAttributes": [
    {
      "targetId": null,
      "targetType": null,
      "value": null,
      "name": "com.amazonaws.ecs.capability.logging-driver.awslogs"
    },
    {
      "targetId": null,
      "targetType": null,
      "value": null,
      "name": "com.amazonaws.ecs.capability.ecr-auth"
    },
    {
      "targetId": null,
      "targetType": null,
      "value": null,
      "name": "com.amazonaws.ecs.capability.docker-remote-api.1.19"
    }
  ],
  "pidMode": null,
  "requiresCompatibilities": [
    "EC2"
  ],
  "networkMode": null,
  "runtimePlatform": null,
  "cpu": null,
  "revision": 8,
  "status": "ACTIVE",
  "inferenceAccelerators": null,
  "proxyConfiguration": null,
  "volumes": []
}
```

Create a service and name it pipelines_demo and associate the task you just created

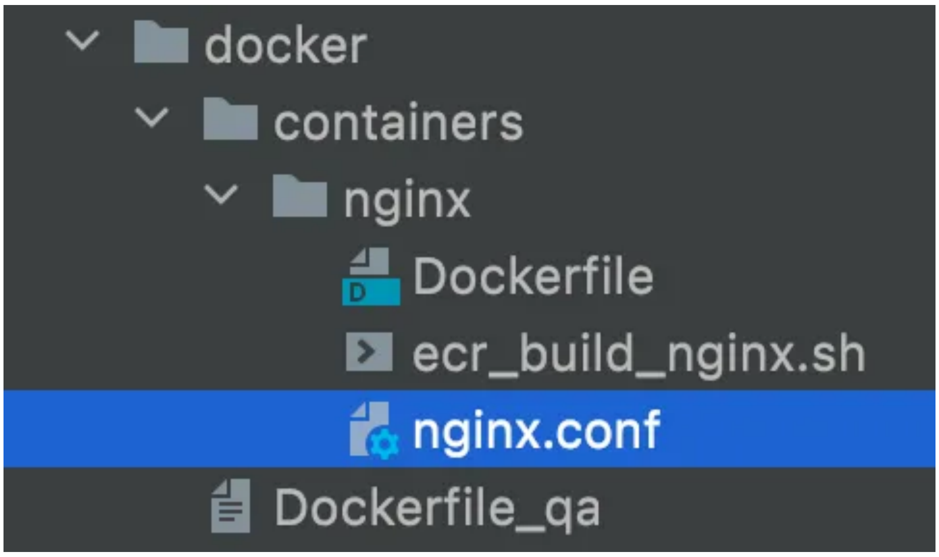


Launch type	FARGATE [✓ EC2] EXTERNAL
	Switch to capacity provider strategy
Task Definition	Family: pipelines_demo_8000 (Enter a value)
	Revision: 2 (latest)
Cluster	QA-2
Service name	
Service type*	[✓ REPLICA] DAEMON
Number of tasks	1
imum healthy percent	100
Maximum percent	200
oyment circuit breaker	Disabled

Create a new repository called Pipelines in GitLab.

Initialise a simple Django application with Python.

Now to set up a Docker image for Nginx



```
docker
  containers
    nginx
      Dockerfile
      ecr_build_nginx.sh
      nginx.conf
  Dockerfile_qa
```

```
#Dockerfile
FROM nginx:1.19.0-alpine

RUN rm /etc/nginx/conf.d/default.conf
COPY nginx.conf /etc/nginx/conf.d
```

#ecr_build_nginx.sh

replace *** as appropriate. You want to use the same name as the repository you created to keep it consistent

```
#!/bin/sh
aws ecr get-login-password --region eu-west-2 | docker login
--username AWS --password-stdin ***.dkr.ecr.eu-west-2.amazonaws.com
# Build the docker container ( stipulating AMD chipset if on a Mac M1
)
docker build  --no-cache --platform linux/amd64 -t pipelines-nginx .
# Tag the completed build in ECR
docker tag pipelines-nginx:latest ***.dkr.ecr.eu-
west-2.amazonaws.com/pipelines-nginx:latest
# Push this build to ECR as latest
docker push ***.dkr.ecr.eu-west-2.amazonaws.com/pipelines-
nginx:latest
```

# nginx.conf

```
# should match name in docker-compose or container in ECR
upstream pipelines-django {
    server pipelines-django:9000;
}

server {

    listen 80;
    client_max_body_size 100M;

    location / {
        proxy_pass http://pipelines-django;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_redirect off;
    }

}
```

Set up Docker to run this application

```
# Dockerfile
FROM python:3.7-slim

RUN apt-get update
RUN apt-get install -y libpq-dev zip
RUN apt-get install gcc -y

COPY . /pipeline
RUN chmod +x pipeline/aws_start.sh
RUN pip install -r /pipeline/dev-requirements.txt
RUN pip install gunicorn
VOLUME /pipeline
WORKDIR /pipeline
EXPOSE 4000

ENV PYTHONUNBUFFERED 1
RUN ["chmod", "+x", "/pipeline/aws_start.sh"]
ENTRYPOINT /pipeline/aws_start.sh
```

Docker will run this entry file

```
#!/bin/sh

# aws_start.sh
python manage.py migrate
python manage.py collectstatic --noinput
python manage.py create_default_superuser
gunicorn -w 3 -b :4000 pipeline.wsgi:application
```

This script will create .env file. You will notice it grabs secret text from AWS systems manager. It will also build and deploy the application to ECS

# ecr_deploy.sh

```
#!/bin/sh
mv .env .env_local

# Copy the QA env to the project root so it is automatically picker
up during docker build
echo -en 'ALLOWED_HOSTS=*\n' >> .env
AWS_ACCESS_KEY_ID=`aws ssm get-parameter --name pipelines-
AWS_SECRET_ACCESS_KEY | jq -r .Parameter.Value`
echo -en 'AWS_ACCESS_KEY_ID='$AWS_ACCESS_KEY_ID'\n' >> .env
echo -en 'AWS_S3_REGION_NAME=eu-west-2\n' >> .env
```

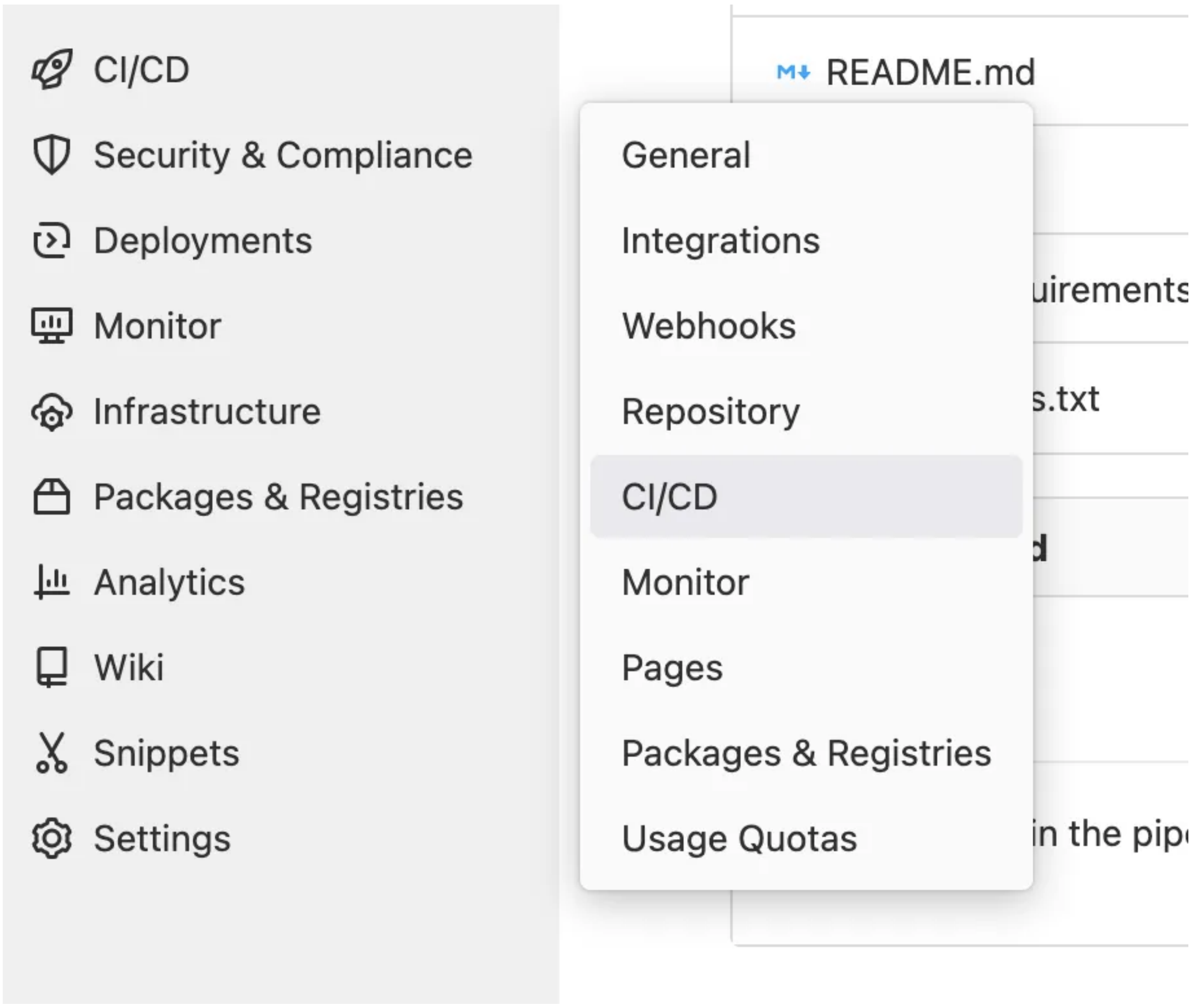
```
AWS_SECRET_ACCESS_KEY=`aws ssm get-parameter --name pipelines-
AWS_SECRET_ACCESS_KEY | jq -r .Parameter.Value`
echo -en 'AWS_SECRET_ACCESS_KEY='$AWS_SECRET_ACCESS_KEY'\n' >> .env
echo -en 'AWS_STORAGE_BUCKET_NAME=pipelines\n' >> .env
echo -en 'BASE_URL=https://pipelines.co.uk\n' >> .env
echo -en 'DATABASE_ENGINE=django.db.backends.mysql\n' >> .env
echo -en 'DATABASE_HOST=pipelines.coz8h02qupfe.eu-
west-2.rds.amazonaws.com\n' >> .env
echo -en 'DATABASE_NAME=pipelines\n' >> .env
DATABASE_PASSWORD=`aws ssm get-parameter --name pipelines-
DATABASE_PASSWORD | jq -r .Parameter.Value`
echo -en 'DATABASE_PASSWORD='$DATABASE_PASSWORD'\n' >> .env
echo -en 'DATABASE_PORT=3306\n' >> .env
echo -en 'DATABASE_USER=pipelines\n' >> .env
echo -en 'DEBUG=0\n' >> .env
echo -en 'DJANGO_SETTINGS_MODULE=zero_nineteen.settings\n' >> .env
echo -en
'EMAIL_BACKEND=django.core.mail.backends.smtp.EmailBackend\n' >> .env
echo -en 'EMAIL_HOST=smtp.eu.sparkpostmail.com\n' >> .env
EMAIL_HOST_PASSWORD=`aws ssm get-parameter --name pipelines-
EMAIL_HOST_PASSWORD | jq -r .Parameter.Value`
echo -en 'EMAIL_HOST_PASSWORD='$EMAIL_HOST_PASSWORD'\n' >> .env
echo -en 'EMAIL_HOST_USER=SMTP_Injection\n' >> .env
echo -en 'ENVIRONMENT=qa\n' >> .env
FEEDBACK_EMAIL=`aws ssm get-parameter --name pipelines-FEEDBACK_EMAIL
| jq -r .Parameter.Value`
echo -en 'FEEDBACK_EMAIL='$FEEDBACK_EMAIL'\n' >> .env
echo -en 'SEARCH_INDEX=pipelines\n' >> .env
SEARCH_URL=`aws ssm get-parameter --name pipelines-SEARCH_URL | jq -r
.Parameter.Value`
echo -en 'SEARCH_URL='$SEARCH_URL'\n' >> .env
SECRET_KEY=`aws ssm get-parameter --name pipelines-SECRET_KEY | jq -r
.Parameter.Value`
echo -en 'SECRET_KEY='$SECRET_KEY'\n' >> .env

# Rename the local dev Dockerfile
mv Dockerfile Dockerfile_local
# Copy the QA Docker file to root ( this one has references in to the
aws_start.sh Entrypoint needed by ECR
cp docker/Dockerfile_qa Dockerfile
# Login to Elastic container registry
aws ecr get-login-password --region eu-west-2 | docker login
--username AWS --password-stdin ****.dkr.ecr.eu-west-2.amazonaws.com
# Build the docker container ( stipulating AMD chipset if on a Mac M1
)

docker build --no-cache --platform linux/amd64 -t pipelines-django .
# Tag the completed build in ECR
docker tag pipelines-django:latest ***.dkr.ecr.eu-
west-2.amazonaws.com/pipelines-django:latest
# Push this build to ECR as latest
docker push ***.dkr.ecr.eu-west-2.amazonaws.com/pipelines-
django:latest
# Update the service on the QA cluster to load the changes
aws ecs update-service --force-new-deployment --service
zero_nineteen_qa --cluster QA-2
```

```
# Delete the Dockerfile with QA settings that was copied into root
rm Dockerfile
# Copy back the Dockerfile intended for local dev
mv Dockerfile_local Dockerfile
# Remove the QA ENVS
rm .env
# Restore the environment variables for local dev
cp .env_local .env
```

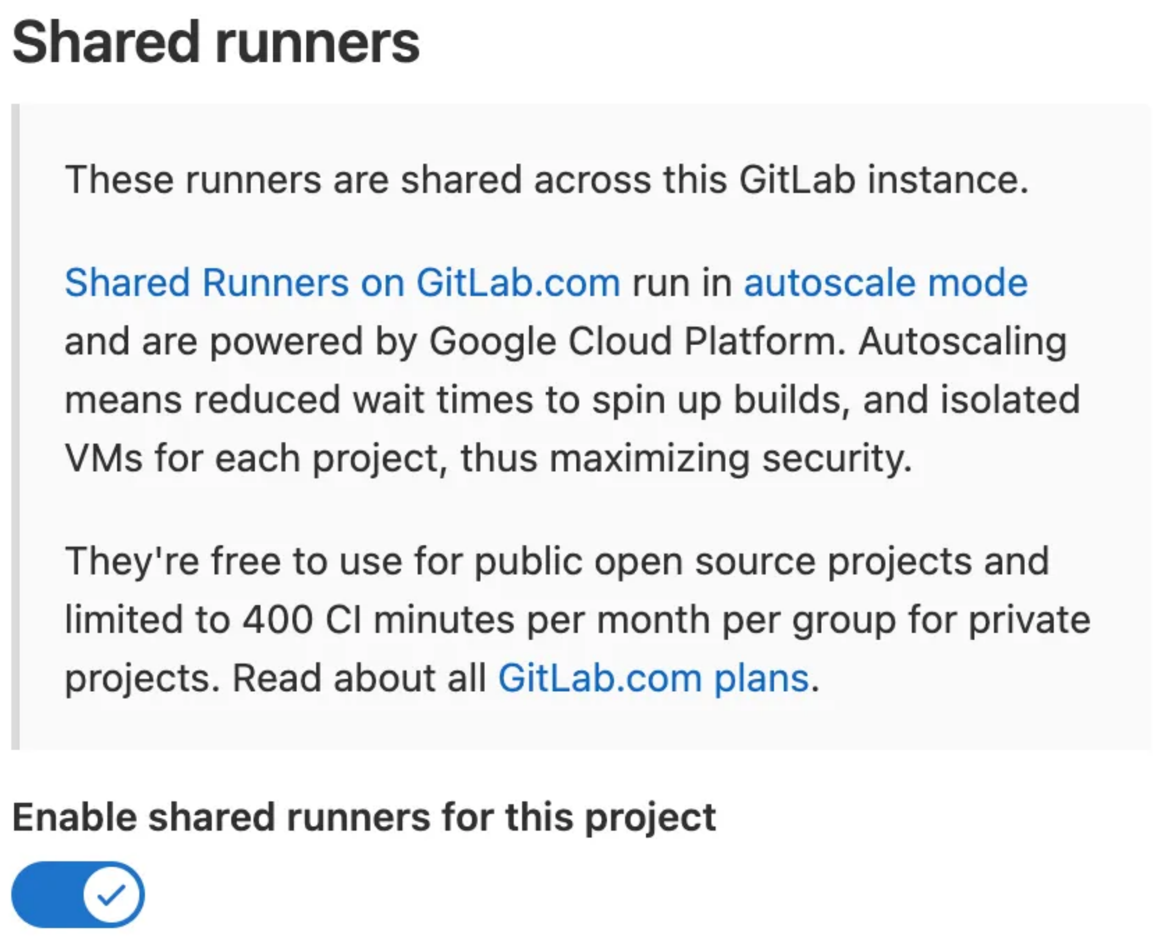
# docker-compose.yml

```
version: '3'
services:

  pipelines_django:
    build: .
    command: ${CMD:-bash -c "pip install -r dev-requirements.txt &&
python manage.py migrate && python manage.py runserver 0.0.0.0:9000"}
    container_name: pipelines_django
    ports:
      - "4000:4000"
    volumes:
      - .:/pipelines
      - ./static:/pipelines/pipelines/static
      - ./media:/pipelines/pipelines/media
    depends_on:
      - pipelines_mysql
    tty: true
    env_file:
      - .env

  pipelines_mysql:
    image: mysql/mysql-server:8.0.23
    command: --default-authentication-plugin=mysql_native_password
    volumes:
      - mysql_data:/var/lib/mysql
    ports:
      - '3306:3306'
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: pipelines
      MYSQL_DATABASE: pipelines
      MYSQL_ROOT_HOST: '%'

  # Use for testing local nginx / QA site
  pipelines_nginx:
    build: ./docker/containers/nginx
    volumes:
      - static_volume:/Users/Documents/projects/pipelines/static/
    ports:
      - 4001:80
    depends_on:
      - pipelines_mysql
      - pipelines_django

volumes:
  mysql_data:
  static_volume:

networks:
  default:
    external:
      name: pipelines_network
```

Set up CI/CD settings in the new repo.



Enable shared runners under runners.



Shared runners

These runners are shared across this GitLab instance.

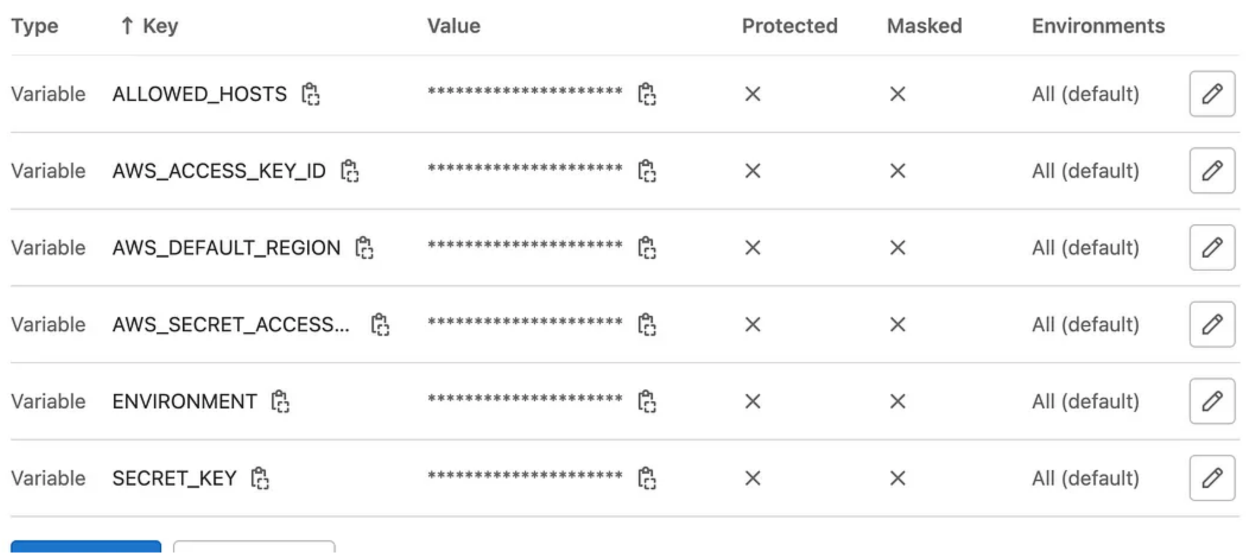
Shared Runners on GitLab.com run in autoscale mode and are powered by Google Cloud Platform. Autoscaling means reduced wait times to spin up builds, and isolated VMs for each project, thus maximizing security.

They're free to use for public open source projects and limited to 400 CI minutes per month per group for private projects. Read about all GitLab.com plans.

Enable shared runners for this project

✓

Under variables I have created the following. you want to add the ones you will use in the GitLab template file in your application `.gitlab-ci.yml`



Type	↑ Key	Value	Protected	Masked	Environments
Variable	ALLOWED_HOSTS	********************	×	×	All (default)
Variable	AWS_ACCESS_KEY_ID	********************	×	×	All (default)
Variable	AWS_DEFAULT_REGION	********************	×	×	All (default)
Variable	AWS_SECRET_ACCESS...	********************	×	×	All (default)
Variable	ENVIRONMENT	********************	×	×	All (default)
Variable	SECRET_KEY	********************	×	×	All (default)

Variables explained

ALLOWED_HOSTS: used for Django

AWS_ACCESS_KEY_ID : I created dedicated key for pipeline deployment

AWS_ACCOUNT_ID: Your AWS Account Id

AWS_REGION: region where you will deploy

AWS_SECRET_ACCESS_KEY: when you create new access key you will get this too

ENVIRONMENT: use for application

SECRET_KEY: used for application

These variables will be passed into the container so the application will have access to them during the pipeline deployment

# .gitlab-ci.yml

```
image: python:latest

services:
  - name: docker:dind
    entrypoint: ["env", "-u", "DOCKER_HOST"]
    command: ["dockerd-entrypoint.sh"]

stages:
  - compile
  - build
  - test

variables:
  DOCKER_HOST: tcp://docker:2375
  DOCKER_DRIVER: overlay2
  DOCKER_TLS_CERTDIR: ""
  MOUNT_POINT: /builds/$CI_PROJECT_PATH/mnt
  REPOSITORY_URL: $AWS_ACCOUNT_ID.dkr.ecr.eu-
west-2.amazonaws.com/pipelines-django
  TASK_DEFINITION_NAME: pipelines_4000
  CLUSTER_NAME: QA-2
  SERVICE_NAME: pipelines
  ARTIFACT_REPORT_PATH: "app/reports/"
  PIP_CACHE_DIR: "$CI_PROJECT_DIR/.cache/pip"

cache:
  paths:
    - .cache/pip
    - venv/

compile:
  stage: compile
  before_script:
    - python -V  # Print out python version for debugging
#     - python -m venv venv
#     - source venv/bin/activate
    - pip install -r dev-requirements.txt
  script:
    - python manage.py makemigrations
  artifacts:
    paths:
      - "*/migrations/*.py"
    expire_in: 1 day
  only:
    refs:
      - merge_requests
    variables:
      - $CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "qa"

build:
  image:
    name: docker/compose:1.25.4
    entrypoint: [ "" ]
  stage: build
  before_script:
    - export IMAGE=$CI_REGISTRY/$CI_PROJECT_NAMESPACE
/$CI_PROJECT_NAME
    - export WEB_IMAGE=$IMAGE/qa_django
  script:
    - apk add --no-cache bash
    - chmod +x ./setup_env.sh
    - bash ./setup_env.sh
    - docker login -u $CI_REGISTRY_USER -p $CI_JOB_TOKEN $CI_REGISTRY
    - docker-compose -f docker-compose-ci.yml build
    - docker tag pipelines_django:latest $WEB_IMAGE:latest
    - docker images
    - [illegible]ker [illegible] $WEB_IMAGE:[illegible]
```

Ci Cd Pipeline Django AWS Gitlab Ci Docker

```
    refs:
      - merge_requests
    variables:
      - $CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "qa"

unittests:
  stage: test
  image: registry.gitlab.com/pipelines/pipelines/qa_django:latest
  script:
    - bash
    - echo $SHELL
    - pwd
    - [illegible]al venv/bin/
    - [illegible]l
    - bash -c "python manage.py jenkins  wellbeing --enable-coverage
--coverage-format=html"
  only:
    refs:
      - merge_requests
    variables:
      - $CI_MERGE_REQUEST_TARGET_BRANCH_NAME == "qa"
```

300 1

Enjoy the read? Reward the writer. Beta

Your tip will go to BoNeZ through a third-party platform of their choice, letting them know you appreciate their story.

Give a tip

Sign up for Top Stories

By Level Up Coding