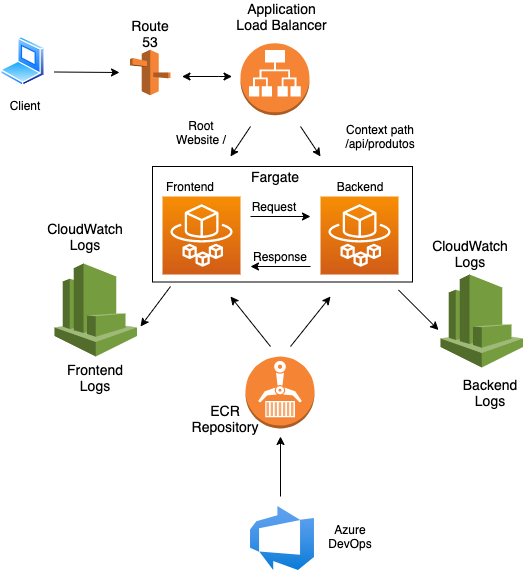
**Projeto**

Para rodar o projeto foi escolhido a Cloud AWS, todos os recursos foram criados em US-EAST-1(Virgínia), abaixo segue um diagrama de representação de como funciona a aplicação:



**AWS**

Os recursos escolhidos para a utilização foram:

**Route53** - para realizar a resolução do nome app.lepeco.com.br para o dns\_name do load balancer.

**Application Load Balancer** - onde possui um listener que redireciona para 2 diferentes target groups:  
- Quando a requisição vem para o site, ele redireciona para o listener de front onde tem a imagem de front como target.

- Quando a requisição é feita diretamente para o backend no path /api/produtos, o listenter através de uma regra onde o context é o path /api/produtos, redireciona para para o target group de backend, onde tem 2 imagens de container rodando.

**ECR** - utilizado com 2 repositório, um para o front e outro para o backend.

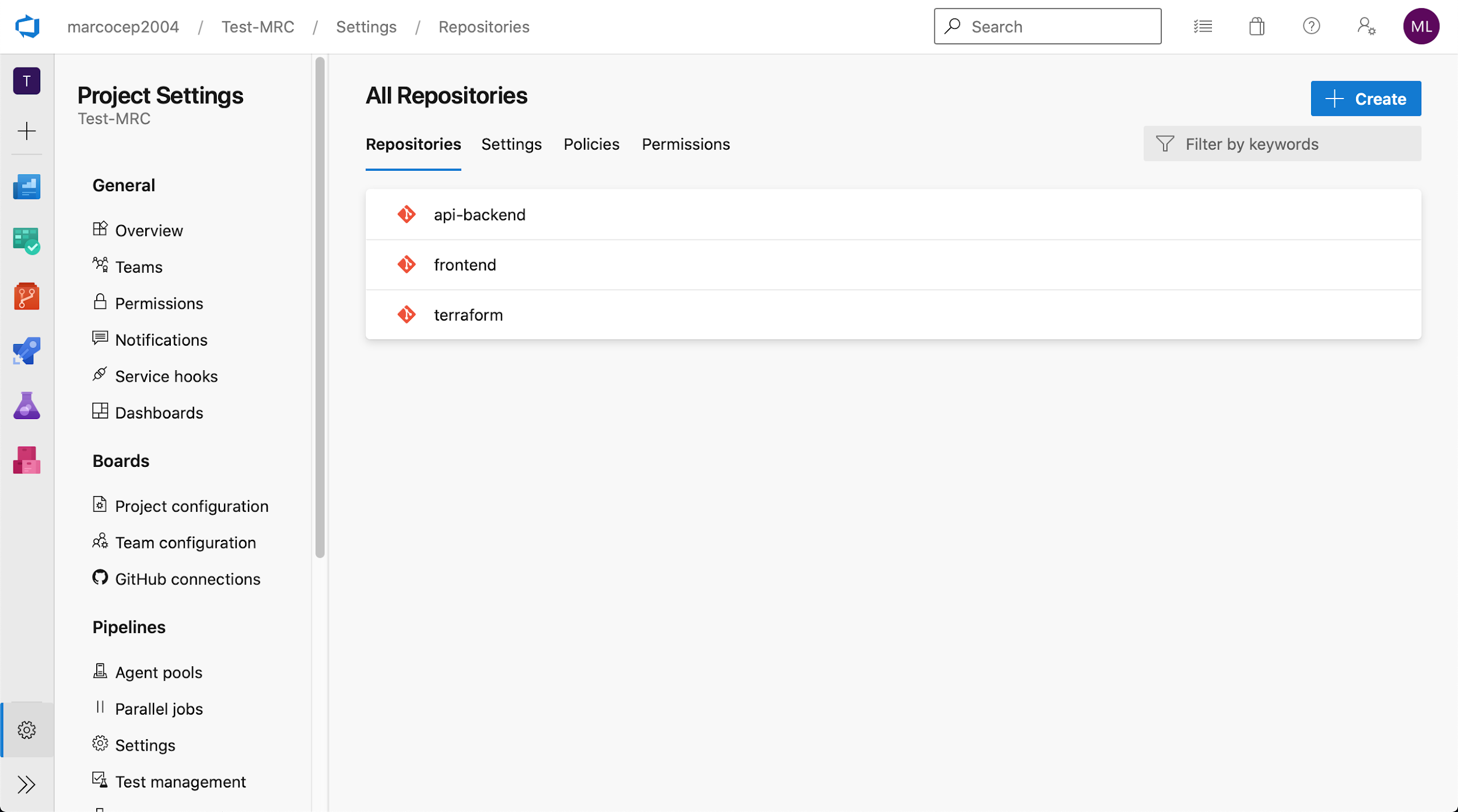
**Fargate** - serviço contido dentro do ECS que realiza a execução de containers sem necessidade de gerenciamento de infra.

**Cloudwatch Logs** - Utilizado para armazenar os logs de execução do container

**VPC** - Foram criadas duas novas Subnets /28 específicas para este projeto.

**Azure DevOps**

Como ferramenta de automação para execução do projeto foi utilizado o Azure Devops. Foi criado um novo projeto chamado Test-MRC, foram configuradas uma Service connection permitindo o acesso da ferramenta da Azure aos recursos da conta de testes na AWS. Foram criados 3 repositórios, 1 para o frontend, 1 para o backend e outro para o terraform:



**Aplicação de exemplo utilizada**

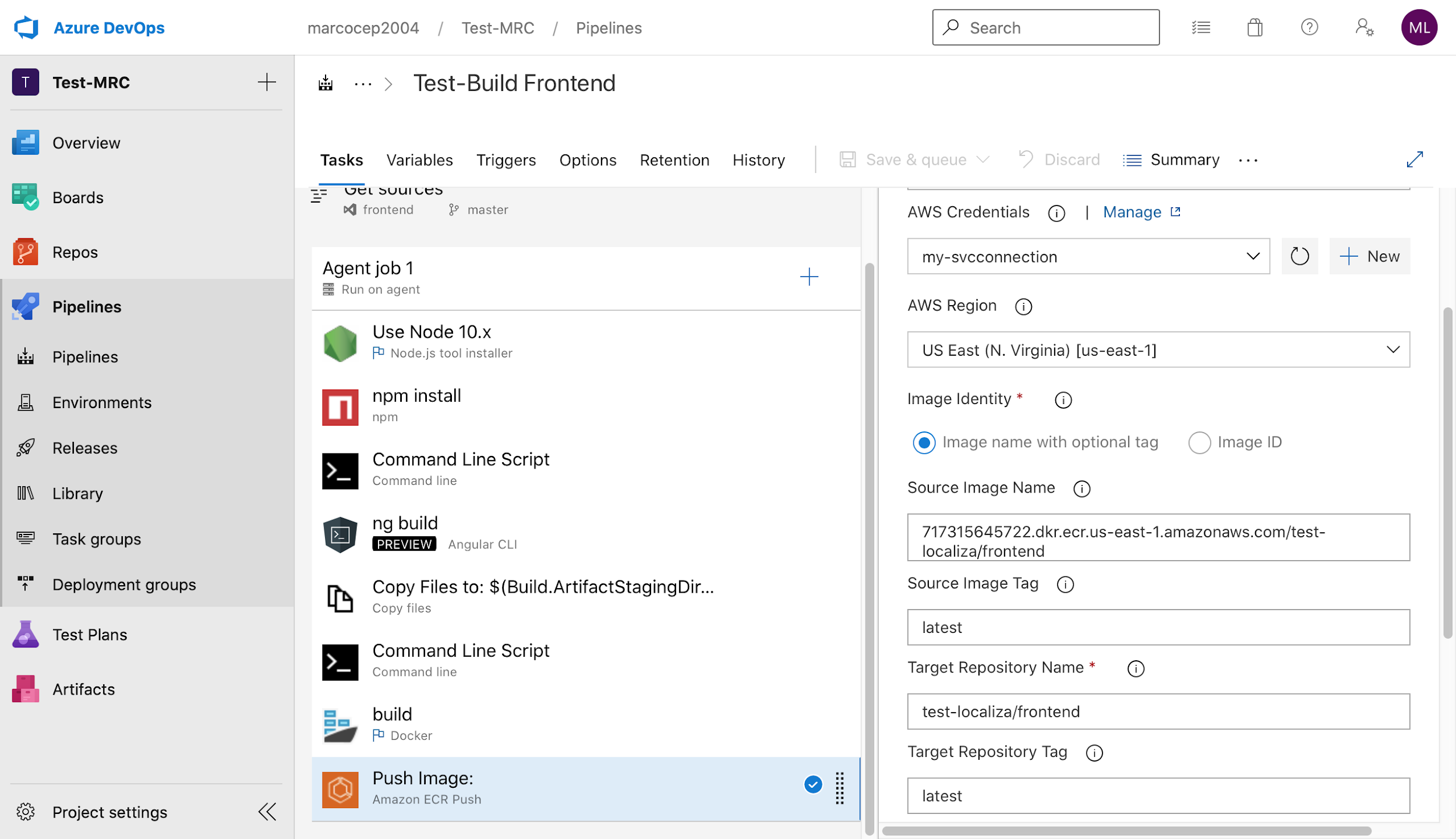
Para demonstrar o funcionamento do projeto foi utilizada uma aplicação que utiliza um frontend usando o framework Angular e o backend utilizando .Net Core, ambas preparadas para rodar em container.

**Fonte: https://medium.com/@renato.groffe/docker-no-back-e-no-front-um-exemplo-com-net-core-e-angular-campus-party-brasil-2019-f15f467d388e**

**Frontend**

Foi criado um build pipeline no Azure DevOps utilizando os plugins que podem ser observados na imagem a seguir, onde é gerado um build da imagem utilizando as configurações contidas no Dockerfile, que está na raiz do repositório, e em seguida é realizado o Push da imagem de container para o ECR da AWS:

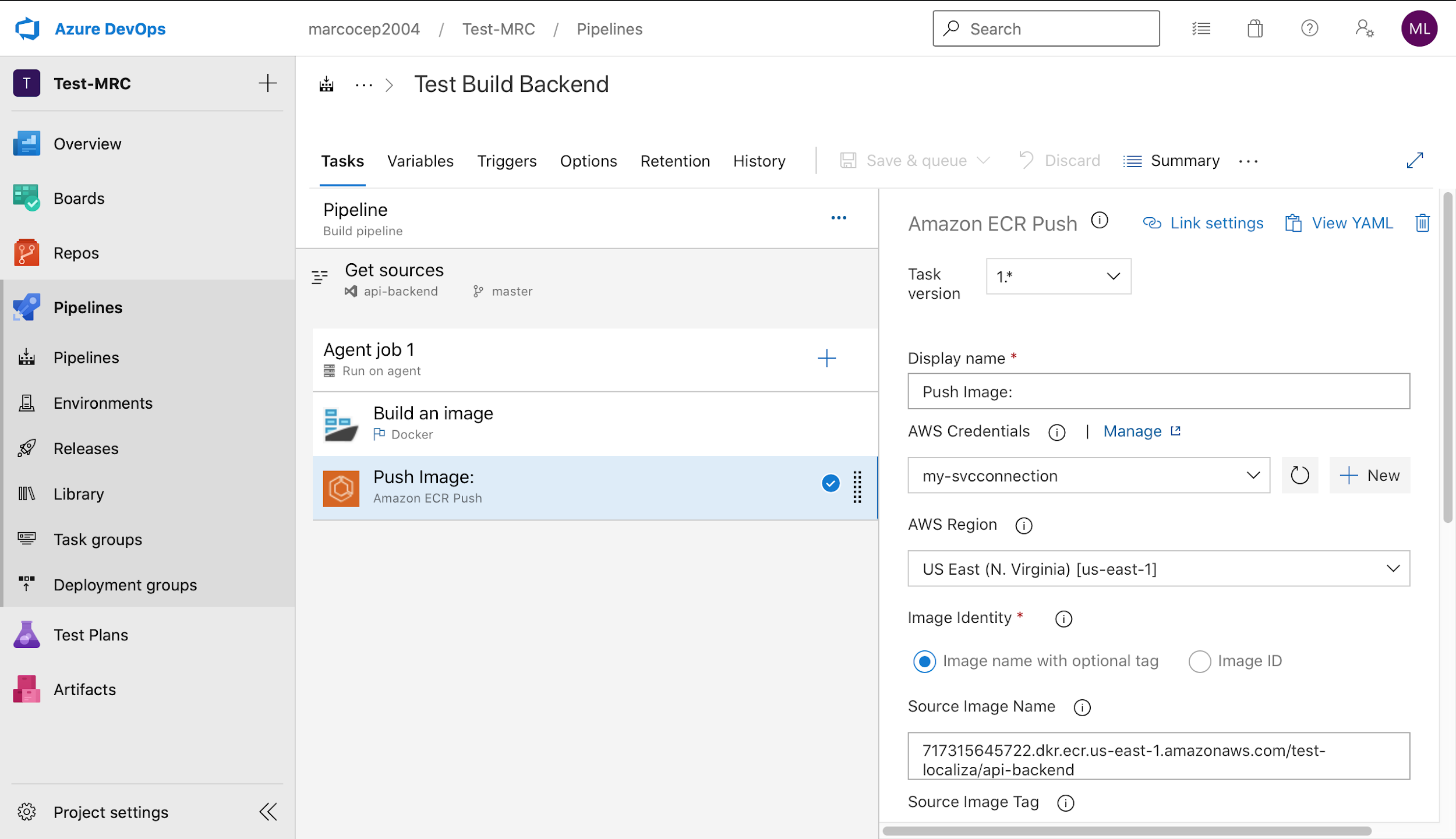




**Backend**

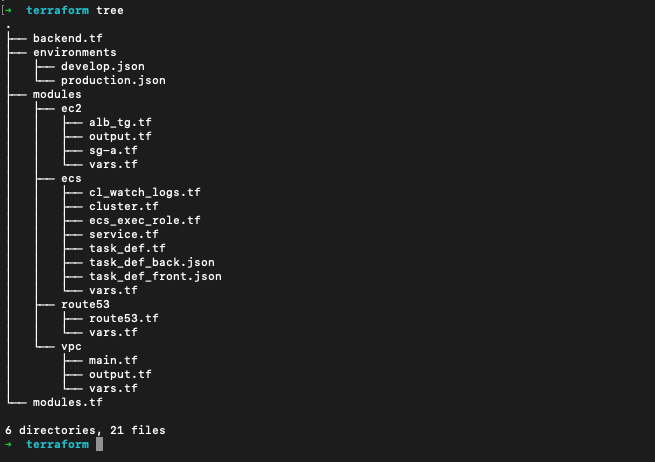
Da mesma forma que o front, o backend também é gerado a partir de um Dockerfile no qual já tem configuradas as informações sobre o build:



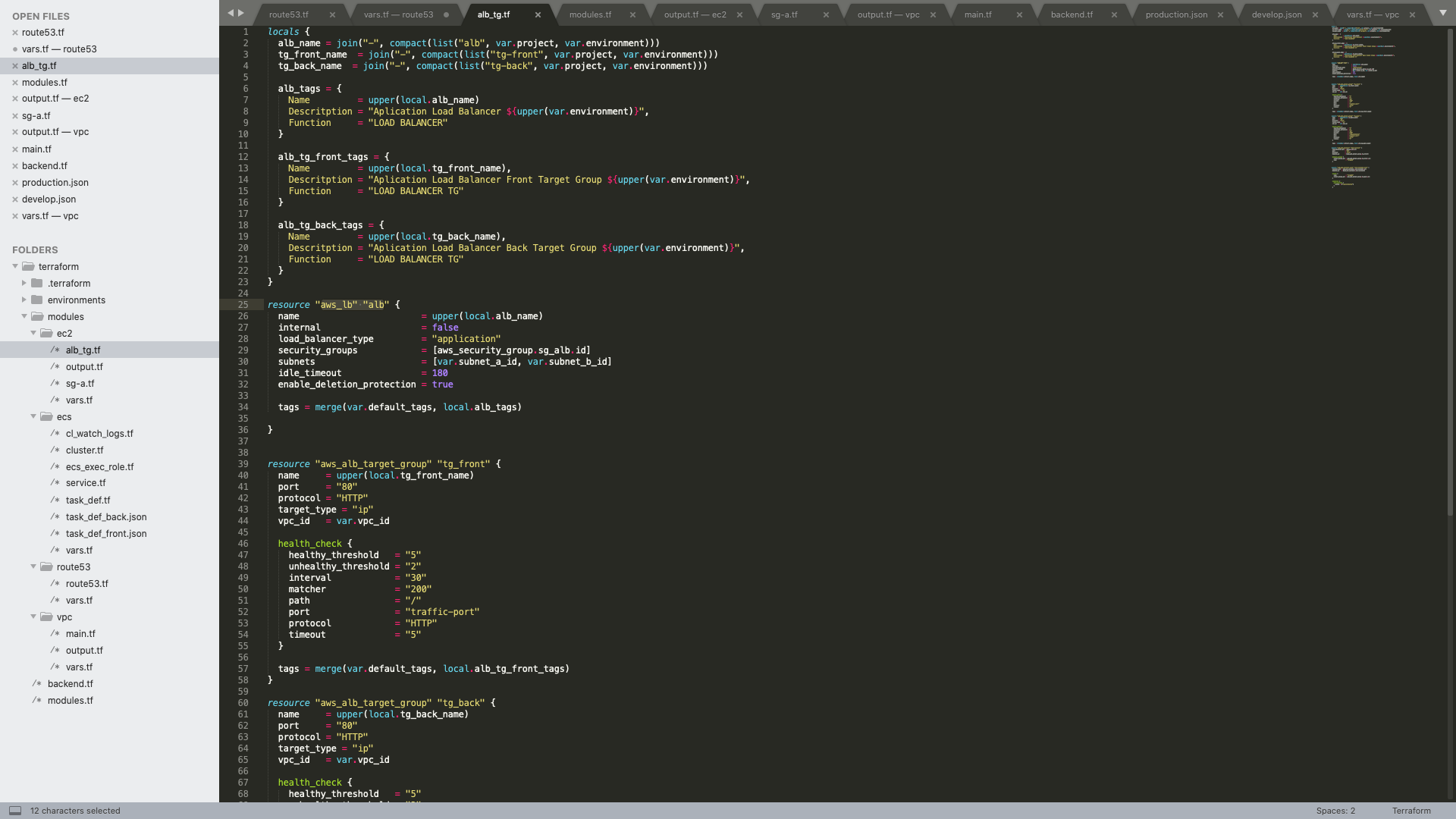


**Terraform**

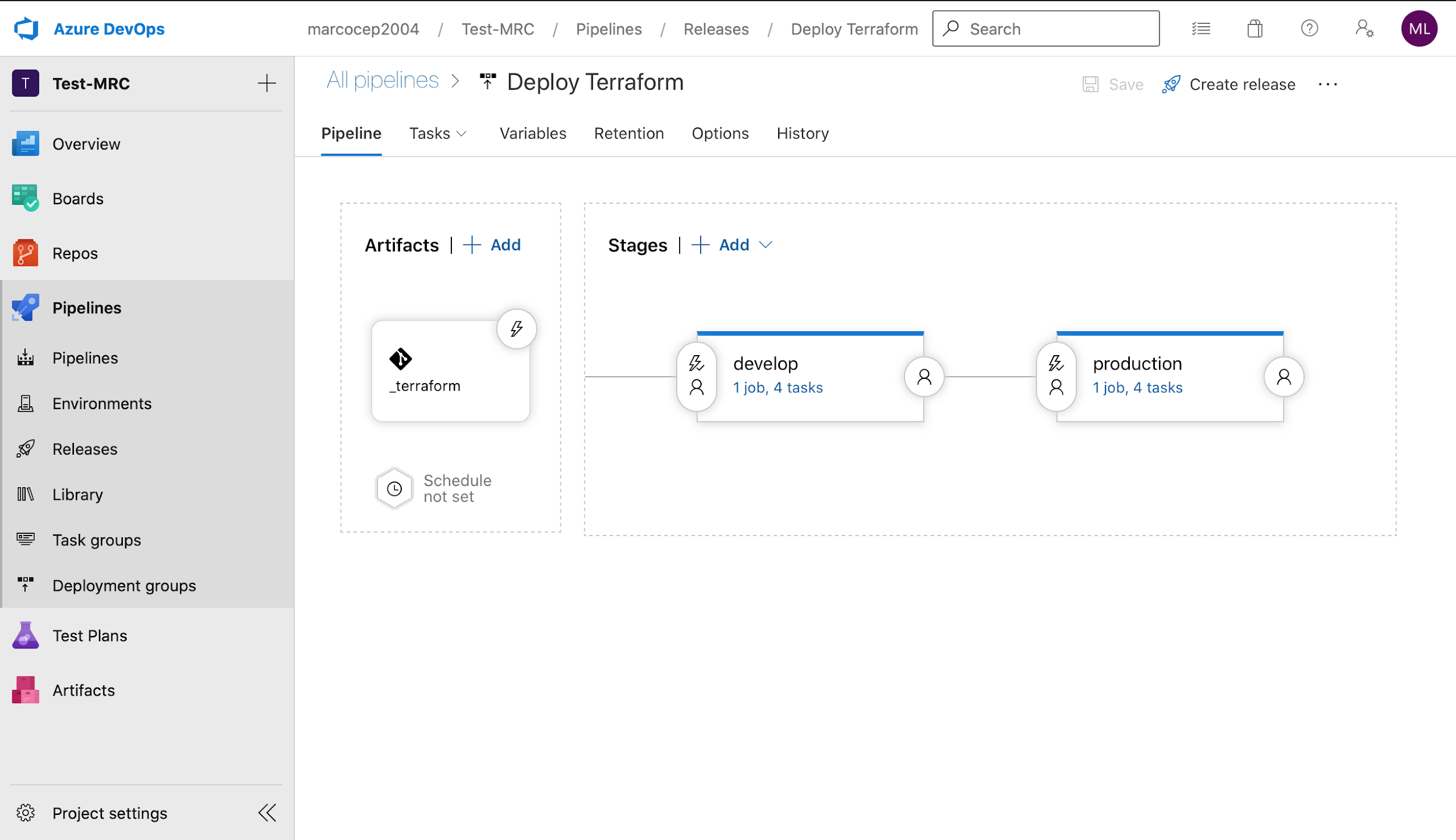
Para orquestração da Infra foi utilizada a ferramenta de Infra-Às-Code Terraform. Para melhor documentação e organização do código foi utilizado o conceito de módulos, e para reaproveitamento de código e controle de ambiente foi utilizado workspace.



O desenvolvimento dos códigos do Terraform foram feitos e testados localmente utilizando o Sublime Text com plugins para o Terraform:

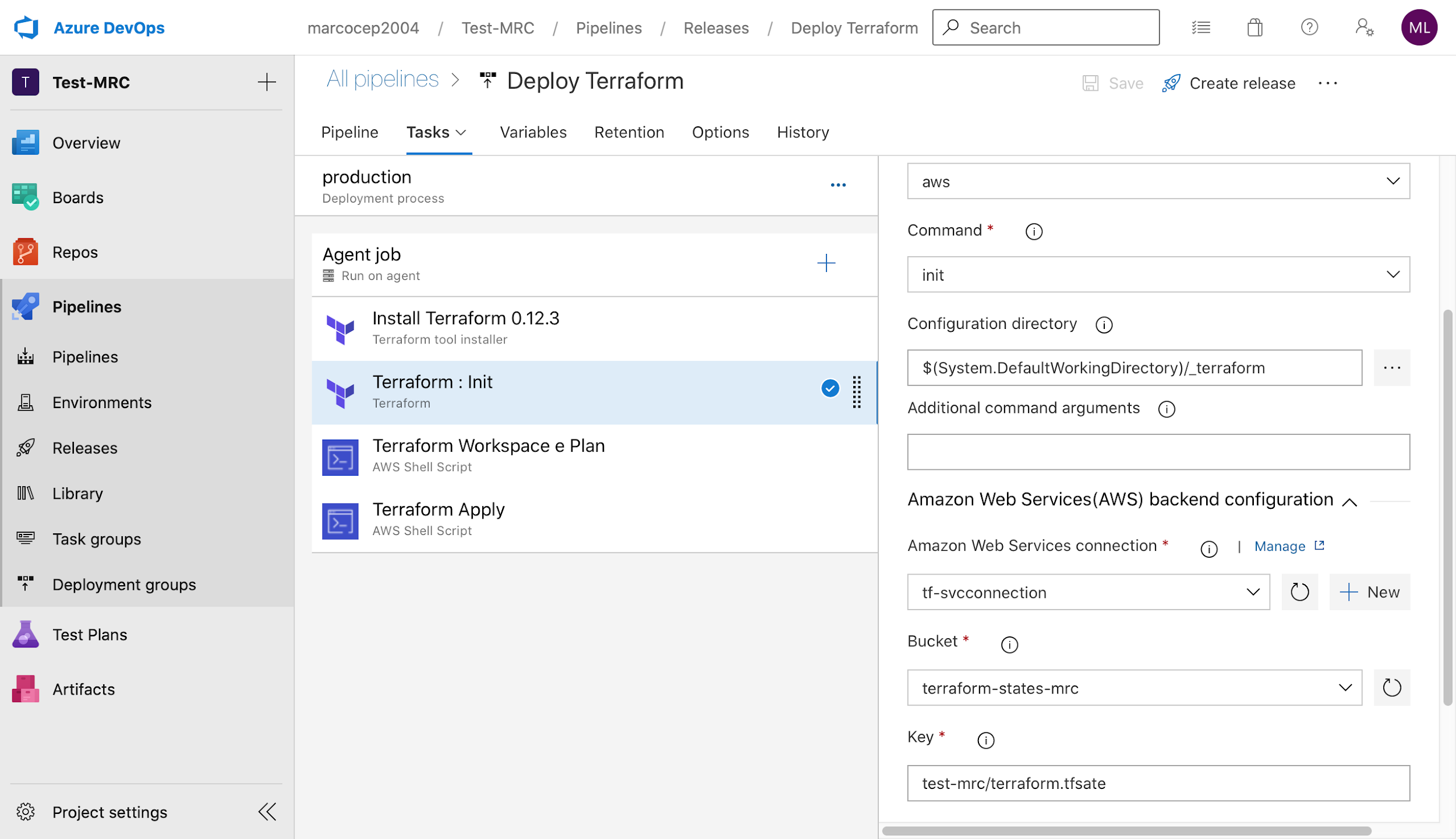


Após a validação foi realizado o commit no repositório terraform do Azure Devops foi criado uma pipeline de execução do Terraform para 2 ambientes, Develop e Production:

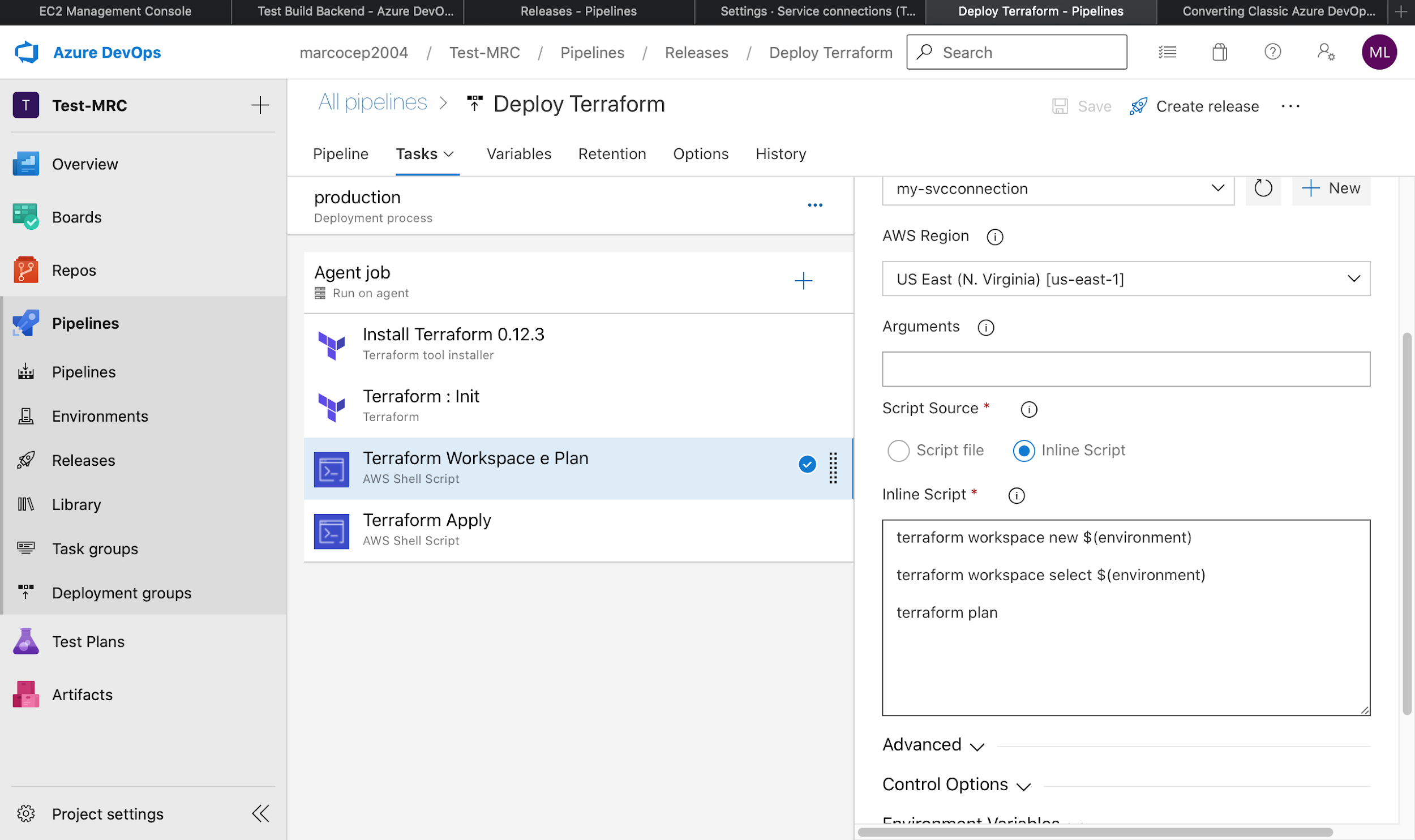


**Deploy do Terraform utilizando Azure DevOps**

Para realizar a inicialização do Terraform foi utilizado um plugin para o mesmo no Azure DevOps, o arquivo de estado do Terraform está sendo salvo em um Bucket do S3.



Por conta do plugin do Terraform para Azure DevOps não suportar Workspace, foi utilizado Shell Script para execução do terraform Plan e Apply:



**Novos release e Update das aplicações**

Após toda construção da Infra na AWS, foi criado release pipelines para novos release ou atualização das imagens em execução em caso de necessidade, foi utilizado AWS Cli na pipeline para terminar a execução com as imagens antigas e subir o serviço com as novas imagens, que são referenciadas pela Tag “latest”:

