Full Stack AWS Development

Project #3

#### Preamble

All the way back in Project #1, the idea for this semesters project was to iteratively build an application, moving from Raw Code (local Node.js server, for instance) to a Docker Deployment, and then onto a Cloud Kubernetes Service.

Project #2 culminated in a full deployment of the stack to Azures AKS platform, supported by ACR image repository. Now, for Project #3, there will be a lift-and-shift of the code, via Docker images, to Amazons Cloud.

#### Objectives

The main objective of this third project is to port Project #2 from Azure AKS to AWS EKS. There were a number of differences in the cloud providers for how this is to be accomplished.

This document includes creating the AWS EKS installation, node creation & AWC Elastic Container Services for holding the Docker images used to create the Kanban-Life app on AWS.

A bit of the code was updated as well, not only for conversion to AWS but also to iteratively improve the application itself. While not exactly a Version 2, it is definitely more robust and portable than it was at the end of project 2.

#### Service Utilization Overview

There are a ton of moving pieces getting an application up and running on any cloud, AWS is no exception – in fact, I found it a bit more challenging to complete Project #3 on AWS than I did with Project #2 on Azure.

The overall view of the services used for this deployment are as follows:

* Stand up a single node EKS area in the Cloud
* Create a Container Repository to hold our application images (ECR)
* Use persistent storage volumes (PVC)
* Utilize a Postgres backend for data management
* Utilize a Node.js/Express/Socket.io web architecture to serve our application across the web
* Use a AWS External Load balancer to expose our application over the public web, while keeping our Postgres DB out of the public eye
* This project is using EKS managed servers, as such, they will be actual EC2 instances, which can also be seen by looking in the EC2 dashboard on AWS
* Finally, a Jenkins container will be intergrated to the Kubernetes service to perform basic tasks on the containers/pods

All files for this project are in GitHub at: <https://github.com/harvash/AWS-full-stack> There are many references to these files simply as the “repo”, this is the starting directory for everything deployed.

#### User setup via IAM

Add user "kanban" to IAM

Login to the AWS console as Root owner ( or IAM user with privileges)

Find IAM in the services search box

Click User > Add User

A screenshot of a social media post

Description automatically generated

Click: Next Permissions

Group Name: kanban-sg

Select AdministratorAccess

Click: Create Group

Click: Tags

kanban = security group permissions

Click: Next Review

Click: Create User

Make note of login info:

A screenshot of a cell phone

Description automatically generated

#### Add tools to your workstation (Laptop or PC)

I will be using a Mac for operations in this project.

Start with installing kubectl (if not already installed) for MacOS

$ brew install kubectl

$ kubectl version --client

Client Version: version.Info{Major:"1", Minor:"16+", GitVersion:"v1.16.6-beta.0", GitCommit:"e7f962ba86f4ce7033828210ca3556393c377bcc", GitTreeState:"clean", BuildDate:"2020-01-15T08:26:26Z", GoVersion:"go1.13.5", Compiler:"gc", Platform:"darwin/amd64"}

Next, add the AWS command line interface

$ curl "https://awscli.amazonaws.com/AWSCLIV2.pkg" -o "/tmp/AWSCLIV2.pkg"

$ sudo installer -pkg /tmp/AWSCLIV2.pkg -target /

$ aws --version

aws-cli/2.0.34 Python/3.7.4 Darwin/19.5.0 botocore/2.0.0dev38

After adding the awscli, it may be helpful to install EKSCTL for AWS K8s command line, although I didn’t need to expressly use this utility during the course of the project.

$ brew tap weaveworks/tap

$ brew install weaveworks/tap/eksctl

$ eksctl version

0.24.0

#### Configure the AWS environment for local

The following steps will setup the local environment to act on the AWS objects.

Login to AWS IAM console and select Users

Select the kanban user and select the Security Credentials tab

Click: Create access key

Click show to see all information and note this down (or screen sot)

A screenshot of a cell phone

Description automatically generated

**Access key ID**

**Secret access key**

AKIA3EB2BJVGH5D3VYWN

wC4HoVbkwyFo90CJU6hLTXS8b7HHi3AZYeGRFVA9

**Use the awscli to configure the environment**

$ aws configure

AWS Access Key ID [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*VYWN]: AKIA3EB2BJVGH5D3VYWN

AWS Secret Access Key [\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FVA9]: 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

Default region name [us-west-2]:

Default output format [text]:

**Create the Virtual Private Cloud**

From the services search, find CloudFormation

This is required for deploying a k8s clusters

From the Create Stack menu, select Create with new resources

Make sure "Template is ready" and "Amazon S3 URL" are selected and add the following URL to the URL box:

<https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-06-10/amazon-eks-vpc-private-subnets.yaml>

Click: Next

Configure the network parameters. I left the defaults in place, as I don't see any apparent network collisions

Click: Next

Tags: kanban = kanban-stack

Click: Next and Create Stack

AWS shows the Stack being created:

A screenshot of a cell phone

Description automatically generated

This will eventually create a lengthy list of resources, once everything is created, the k8s cluster can be created.

**Create EKS cluster:**

From the Services search, find EKS

On the far right, enter "kanban-eks" and click, next step.

Under Cluster Service Role, right click IAM console and open in new tab/window, this allows creating a new EKS manager role.

In IAM:

Click: Create role and choose EKS from list, as well as EKS-Cluster for use case.

Click: Next permissions and then click: Tags

Tags: kanban = kanban-iam

Click: Next Review

Name the role: kanban-iam

Click: Create role, after it returns to the IAM dashboard, close the tab/window and return to the EKS configuration tab/window.

Click the refresh icon and select the kanban-iam role

Leave Secrets encryption off for now and add Tags

kanban = kanban-cluster

Click: Next

In VPC info, click the dropdown box and select the "kanban-stack-vpc"

In the Security groups dropdown, select "kanban-stack-ControlPlan-SeurityGroup"

In Cluster endpoint access, select "Public and private"

Click: Next

Leave Control Plane Logging disabled for now and click: Next

Review the settings and click: Create

The cluster is now being created:

A close up of a logo

Description automatically generated

A screenshot of a cell phone

Description automatically generated

**Setup Elastic Container Registry (ECR)**

Probably the most necessary piece of this project is using the ECR to store the Docker images that are used in constructing the Kubernetes deployments.

From the services search, enter Elastic container registry

Click: get started

Name the repository kanban-ecr

Click: Create repository

**Extra steps for using ECR with Docker on local workstation**

There is an add-on helper to configure Docker to interact with ECR without needing to get tokens.

For MacOS:

$ brew install docker-credential-helper-ecr

Configure the helper:

Add the following to the ~/.docker/config.json file

"credHelpers": {

"764625898828.dkr.ecr.us-east-1.amazonaws.com": "ecr-login"

}

There were instances where the credential helper did not work, under these circumstances, a manual login was performed:

$ aws ecr get-login-password --region us-west-2 | docker login --username AWS --password-stdin 764625898828.dkr.ecr.us-west-2.amazonaws.com

#### Review the Code

The code in the repo is organized as follows:

Web application - ./server/app/Dockerfile

Postgres DB - ./compose-postgres/Dockerfile.pgdb

PgAdmin - ./compose-postgres/Dockerfile.pgadmin

Jenkins - ./jenkins/Dockerfile

*00\_namespace.yaml*

This configures the namespace to deploy the application into. Namespaces are helpful to organize deployments and keep things well ordered.

*01\_deploy\_cloud\_services.yaml*

This configuration sets up the services used to connect to the various applications. It also maps the URI to a port on each deployment

02\_deploy\_backend.yaml

This file deploys a statefulset and uses Persistent Volume Claims so that data is not lost when a node crashes or maintenance is performed.

Statefulsets are a unique deployment that are designed to be well ordered and behave in a certain manor. They are extremely useful when used with a scaling application. The statefulset used in this project has only one member, however that could be updated to deploy multiples of the application.

Taking the PostGres DB as an example, a statefulset could be built to have a master container (read/write) and many reader containers (read only). The service is headless and would determine traffic endpoints via the web call.

*03\_deploy\_frontend.yaml*

This yaml file is used to create the PgAdmin container. PgAdmin is used to “look into” the database, as well as to perform any db-centric actions – like adding a table or truncating data.

*04\_deploy\_web.yaml*

If the database is the heart of the application, the web container is the brains. This container will spin up a Node.js server, integrate with PostGres API’s and display the web pages related to the application. It is also responsible for taking in user input (creating a new board) and shipping that data off to the Postgres backend – all while updating connecting clients in real time.

*05\_deploy\_jenkins.yaml*

Jenkins is worker bee used to keep the flow of Continous Development and Continous Intergration moving along. This project will create a simple job that changes the port used to access PgAdmin. As the project is developed, more jobs can be added to help maintain and upgrade the application.

As a bonus, and using good CI/CD architecture, Git hooks will be used to trigger jobs on a commit to the repo. The simple job demonstrated here updates the Services file when Jenkins receives notification from a GitHub commit and automatically rebuilds & deploys the updated configuration

**Pushing Images to the ECR**

Once the ECR and EKS are provisioned and working, the local docker images need to be pushed to the AWS repository. The Docker files for the project are in the GitHub repo AWS-full-stack/kanban-life under their respective applications, as follows:

The images can all be built via the customary *docker build* command

Example:

$ cd kanban-life

$ docker build -t kanban-pg -f ./compose-postgres/Dockerfile.pgdb ./compose-postgres

After the Dockerfiles are built (4 total), the images need to be tagged for the ACR and pushed to AWS.

Due to the way ECR is defined, the ECR repo name MUST be the tag used to push the image. In the case where there are multiple images for a project, like kanban-life, the tag can be modified to indicate with application the image contains.

Example:

$ docker tag kanban-pg:latest 764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr:kanban-pg

As in the example above, the Docker image for the Postgres DB is pushed to the kanban-ecr repo with a tag of "kanban-pg". Doing this allows the user to see which images have what contents.

After properly building and tagging the images, these are images marked for ECS use:

$ docker images|grep kanban-ecr

764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr kanban-pg

764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr kanban-jenkins

764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr kanban-pgadmin

764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr kanban-web

Pushing the Images

$ docker push 764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr:kanban-jenkins

$ docker push 764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr:kanban-pg

$ docker push 764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr:kanban-pgadmin

$ docker push 764625898828.dkr.ecr.us-west-2.amazonaws.com/kanban-ecr:kanban-web

**Setting Up the EKS**

There was a large disconnect at this point between the various entities that setup up the cluster and security profiles needed for access. Primarily, the user (kanban) was setup as an administrative user, however the role (kanban-iam) was used to build the cluster. These entities did not have a relationship, and as a result, there were errors trying to use *kubectl* and *apply* changes. The following steps were used to bring everything into alignment and allow use of *kubectl*, *awscli* and successfully administer the EKS instances.

**Updating kube config with yaml**

AWS provides a config-map for this process. It can be found in the Azure\_full\_stack/AWS-full-stack directory.

The file was obtaining by curl from Amazon:

$ curl -o aws-auth-cm.yaml <https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-06-10/aws-auth-cm.yaml>

Troubleshooting

-------------------------

<https://aws.amazon.com/premiumsupport/knowledge-center/amazon-eks-cluster-access/>

aws configure - use root creds -- allowed kubectl commands

edit was-auth-cm.yaml to mapUsers kanban (using ARN)

$ kubectl apply -f aws-auth-cm.yaml

Update aws configure - use kanban creds

--------------------------

**Setup NODEs (AWS EKS Compute) using Managed Nodes**

Managed Nodes are used for this deployment to allow access to public subnets, use EBS storage and other services not allowed by other types (eg Fargate). Managed Nodes will spin up (and charge for) EC2 instances for supporting the nodes.

Each node should have a separate role to prevent cross-cluster authentication where not intended.

**Add the role**

In the Services search bar, find IAM

Select Roles, and click: Create role

Choose the EC2 common use case, click: Next permissions

Add the following roles:

AmazonEKSWorkerNodePolicy

AmazonEKS\_CNI\_Policy

AmazonEC2ContainerRegistryReadOnly

Click: Next Tags

kanban = kanbanNodeRole

Role name: kanbanNodeRole

A screenshot of a social media post

Description automatically generated

Click: Create role

**Launch the Managed Nodes**

In the Services search, find EKS and click on Amazon EKS > Clusters

Select the cluster to launch nodes for: kanban-eks

Select Compute tab and click: Add Node Group

Name: kanban-nodeGroup

Node IAM Role: kanbanNodeRole

Leave Subnets as default

Ensure Allow remote access is on

For SSH key pairs, right click EC2\_console and open in new window/tab

Click: Create key pair

Name: kanbanNodeSSH

format: pem

Add Tag: kanban = kanbanSSHkeys

Click: Create key pair

A download dialog should open to download the keys to local workstation. If these are placed in a git directory, the file names should be added to .gitignore so they are not uploaded to repository

Close the tab/window and return to the configuration page. Refresh the SSH key selection and choose the kanbanNodeSSH keys

Add Tags for Kubernetes: kanban = kanban-life

Add Tags for EKS: kanban = kanban-nodes

Click: Next

AMI type: Amazon Linux 2(AL2x86\_64)

Instance Type: t3.micro (This is a free tier EC2)

Disk Size: 20G

Click: Next

For Group size, leave all entries at "2"

Click: Next and click: Create

Now from the local workstation, watch the nodes spin up

$ kubectl get nodes --watch

There will be a delay while the nodes spin up, followed by output like "Not Ready" then, finally, "Ready"

Once node deployment is complete, view the status:

$ kubectl get nodes

NAME STATUS ROLES AGE VERSION

ip-192-168-250-174.us-west-2.compute.internal Ready <none> 63s v1.17.7-eks-bffbac

ip-192-168-57-230.us-west-2.compute.internal Ready <none> 63s v1.17.7-eks-bffbac

#### Application Deployments

Change directories in the repo to the k8s directory:

**Create a namespace to keep everything organized**

$ kubectl apply -f 00\_namespace.yaml

namespace/kanban-life created

**Set new Namespace to the context for EKS**

$ kubectl config view

…

contexts:

- context:

cluster: arn:aws:eks:us-west-2:764625898828:cluster/kanban-eks

user: arn:aws:eks:us-west-2:764625898828:cluster/kanban-eks

name: arn:aws:eks:us-west-2:764625898828:cluster/kanban-eks

…

$ kubectl config set-context kanban-life --namespace=kanban-life **\**

--cluster=arn:aws:eks:us-west-2:764625898828:cluster/kanban-eks **\**

--user=arn:aws:eks:us-west-2:764625898828:cluster/kanban-eks

$ kubectl config use-context kanban-life

Change into the k8s directory of kanban-life and apply each deployment.

At this point, it is a good idea to setup an alias for kubectl so that it can have a short identifier

$ echo “alias k=’kubectl’ >> ~/.bashrc (or .bash\_profile, depending on OS flavor)

**Services**:

$ k apply -f 01\_deploy\_cloud\_services.yaml

**Applications**:

$ k apply -f 02\_deploy\_backend.yaml

$ k apply -f 03\_deploy\_frontend.yaml

$ k apply -f 04\_deploy\_web.yaml

Now check that all pods and services are up and running, as well as noting the External IP for the Web & PgAdmin applications

$ k get svc

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE

jenkins LoadBalancer 10.100.188.243 a57020322736d49b9a95a88c2006eace-1202519709.us-west-2.elb.amazonaws.com 8080:30540/TCP 69m

kanban-web LoadBalancer 10.100.32.38 a7c07dfccb26a419c8e8a2575e1a72b9-1612158856.us-west-2.elb.amazonaws.com 4001:30917/TCP 69m

pgadmin LoadBalancer 10.100.158.134 a242571399c16487dae265bb82d962fe-1820973048.us-west-2.elb.amazonaws.com 80:31341/TCP 69m

postgres ClusterIP None <none>

$ k get pods

NAME READY STATUS RESTARTS AGE

kanban-web-5d6754d8cc-w8bs2 0/1 ContainerCreating 0 3s

pgadmin-6d56f55455-ndkjs 1/1 Running 0 10s

postgres-0 1/1 Running 0 16s

External IP’s highlighted in yellow and ports highlighted in green, are for web access to the Application & PgAdmin tool

#### Testing the applications

**PgAdmin Tool**

Navigate to the External IP:port address for PgAdmin.

Create a connection to the PostGres Database

Right click Servers > Create > Server

Under General tab, name the server ‘Kanban’

Fill in the remaining fields per the below screenshot

A screenshot of a cell phone

Description automatically generated

Once the server is added, the database objects can be seen using:

Servers > Kanban > Databases > pgkanban > Schemas > pgkanban > Tables

The Kanban\_list table holds the data for the Kanban boards displayed in the web application.

**Kanban-Life Application**

Navigate the External IP:port for the web application

A screenshot of a cell phone

Description automatically generated

The starting board prepopulated in the Database is shown, along with number of connected users.

Additional browsers may be opened on any device and the connection will update, as well as displaying the current boards in the Database

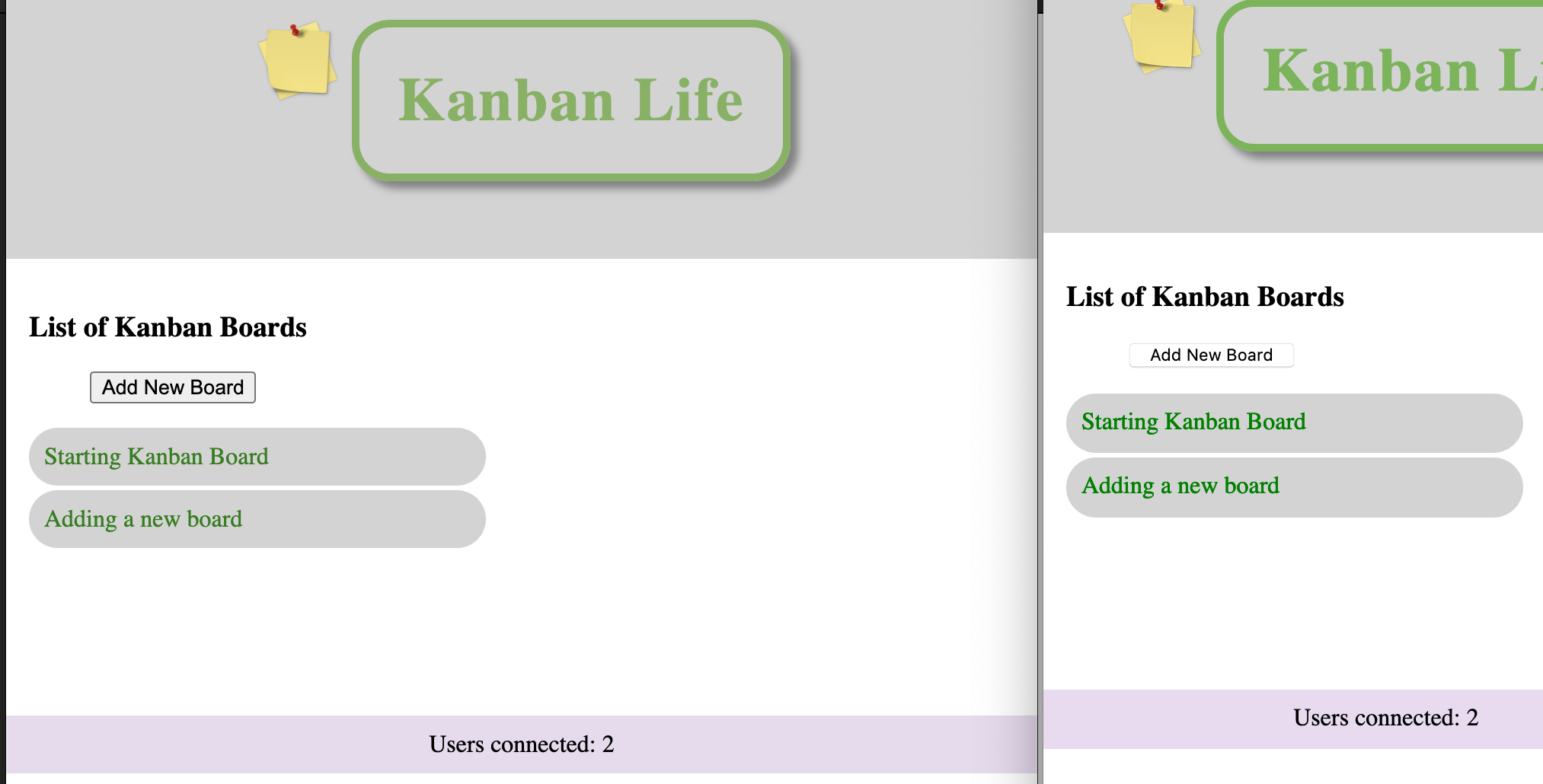
A screenshot of a cell phone

Description automatically generated

Adding a board vi “Add New Board” will update the Database table, as well as any currently connected clients

A screenshot of a cell phone

Description automatically generated



Updated database table

A screenshot of a social media post

Description automatically generated

REFERENCES

<https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-mac.html>

<https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html> <https://github.com/awslabs/amazon-ecr-credential-helper>

<https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html>