



Balai Pengembangan Talenta Indonesia
Pusat Prestasi Nasional
Kementerian Pendidikan, Kebudayaan, Riset, dan Teknologi

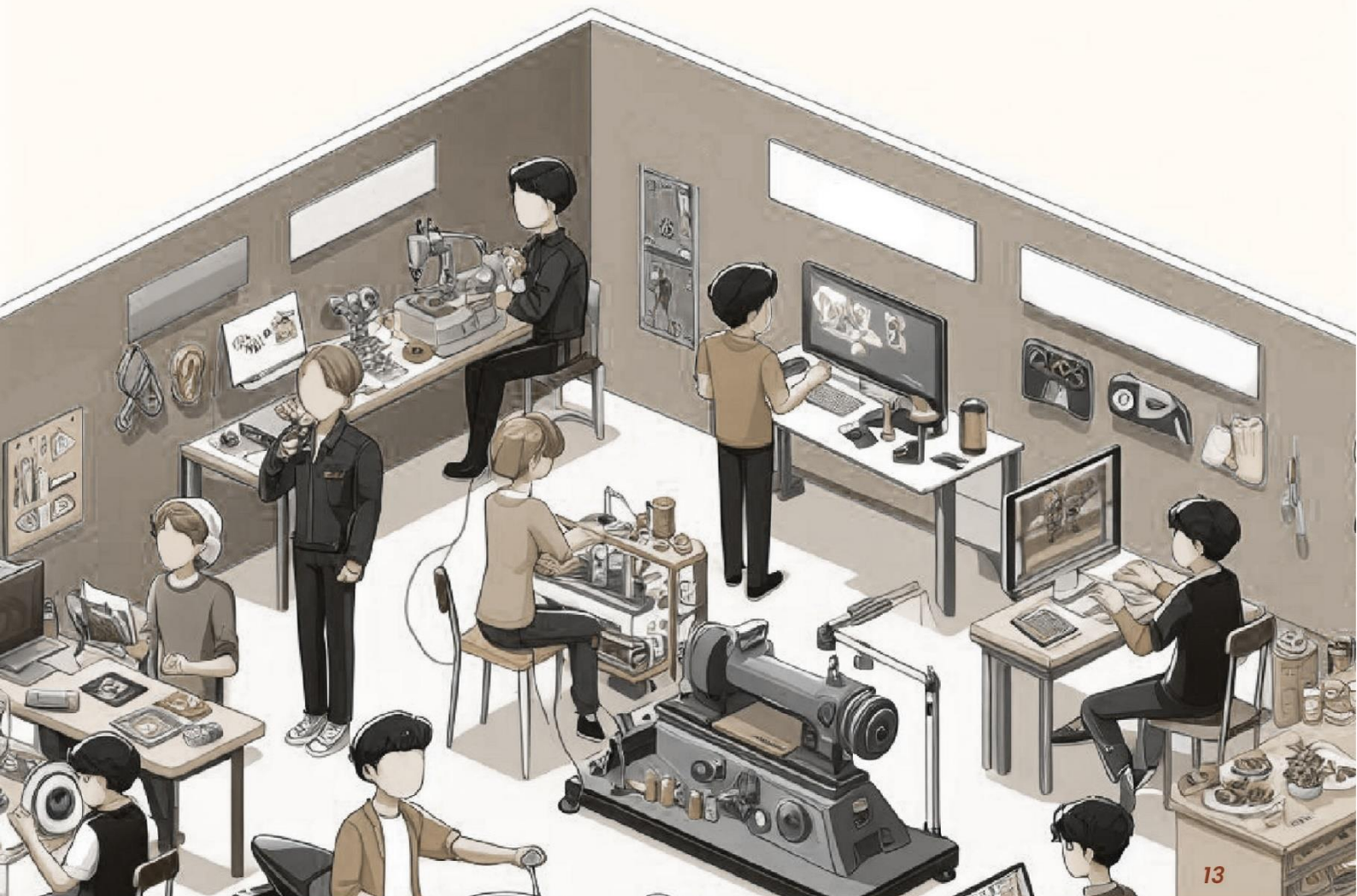
**MERDEKA
BELAJAR**



SMK

Soal Lomba Kompetensi Siswa Nasional 2024

Komputasi Awan
(Cloud Computing)



13

MERDEKA BERPRESTASI
Talenta **Vokasi** Menginspirasi

Description of Project and Tasks

This module is 3 hours

Speaks company is a company that divides surveys and votes. The current application is containerised and runs on elastic container service(ecs). The company is going one step further by Adopting DevOps Metedology. All Speaks Company application environments run on Amazon Web Services (aws). They want you as a DevOps Engineer to build the architecture for Proof of Concept (PoC) for architectures.

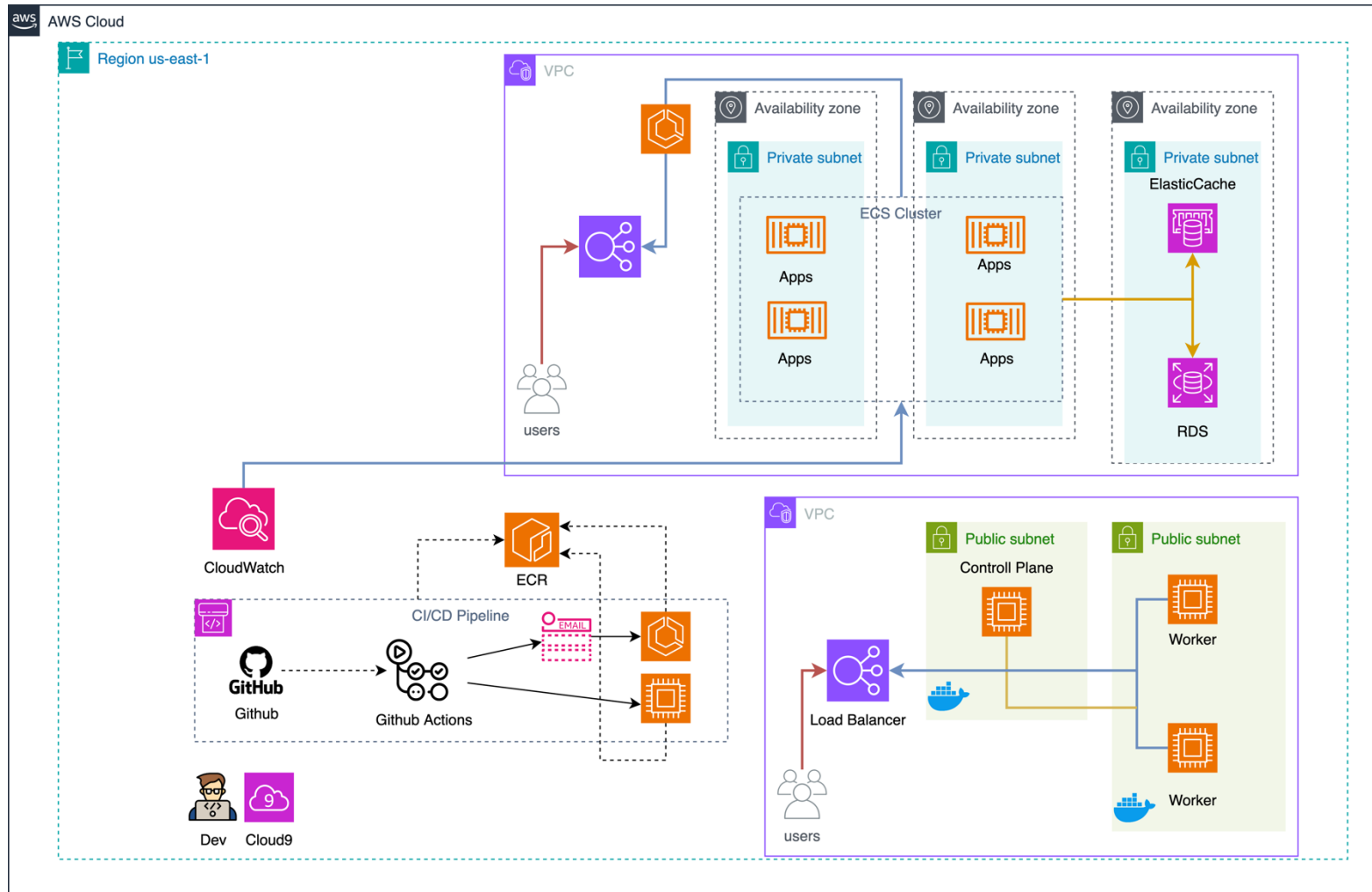
Task

1. Make sure user can login into AWS Academy console
2. Read the documentation thoroughly (Outlined below).
3. Please view the architecture in the **architecture section**
4. Please reach each item in **technical detail** carefully
5. Please reach each item in **application detail** carefully
6. Prepare GitHub account.
7. Go to AWS Academy Learner Lab and Launch Lab
8. Download key pair in AWS Details and Take note AWS CLI credentials
9. You don't need to create a new user or role, just use the existing role, LabRole.
10. Set up the VPC with their respective subnet and security group configurations.
11. Setup initial support services like cloud9, load balancing, database, share storage, and memory data store.
12. Configure any server dependencies as outlined in the **technical details**.
13. Setup and Configure Elastic Container Service Platform for production environment. The ECS configuration detail in the **Elastic Container Service – Service details** section.
14. Create Pipeline according to instruction given in **Deployment Process Details** section and deploy application.
15. Configure necessary application monitoring and metrics in Cloudwatch.

Technical Details

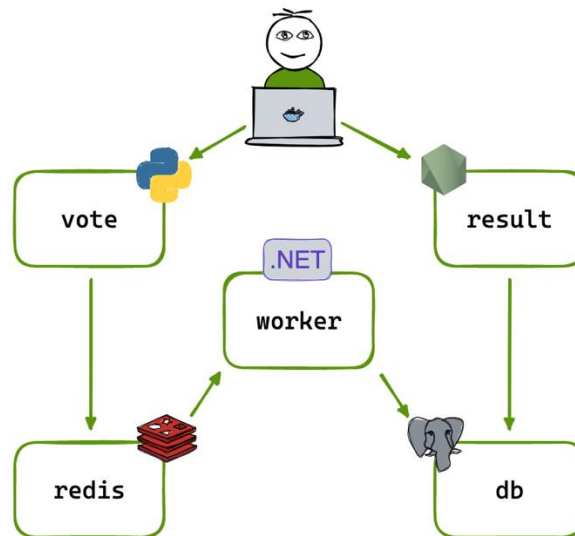
1. The goal of this project is to build automation deployment for microservice application
2. The App is calling **Voting App** and source code is in the same repository, you must fork from the Github source to your GitHub account. Please read the **Application section details**.
3. The Voting app has three services with vote service as the voting page, result service as the page displaying the voting results and worker service as the service that writes data from redis to the database.
4. Application will deploy in 2 environment Development and Production, Development environment will use docker swarm cluster and Production will deploy in Elastic Container Service(ECS)
5. Application must be deployed automatically and triggered by GitHub changes. Be it cluster development or production.
6. You Should be creating an additional security group following in the security group Service details section.
7. The base OS that has been chosen is **Amazon Linux**(<https://aws.amazon.com/amazon-linux-ami/>). This distribution was selected for its broad industry support, stability, availability of support, and excellent integration with AWS.
8. The load balancer will manage and establish the application load.
9. Developer and Operation only access instance or cluster from cloud9.
10. Ensure all the resources build and configured in **us-east-1** region

Architecture



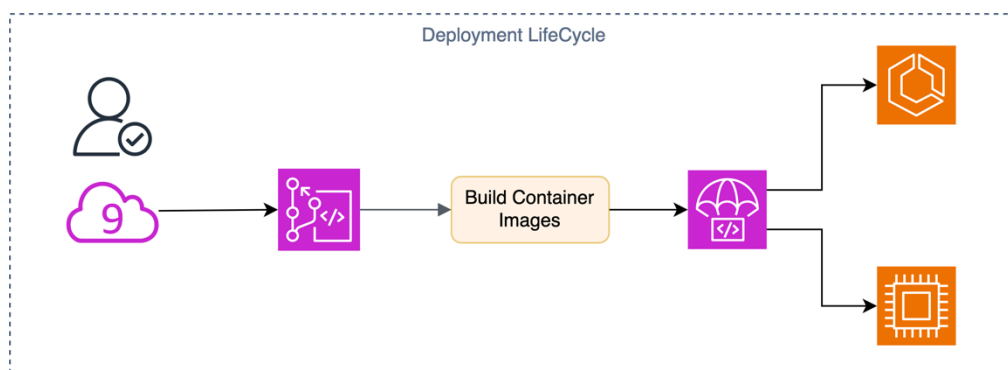
Application Details

The voting Apps was developed by dockersamples. The voting Apps consists of 3 main applications and 2 supporting components. 3 main application is **vote app** was developed with Python, **result app** was developed with NodeJS and **worker app** was developed with .NET. In order to run well, The voting apps require 2 supporting components, namely postgresSQL and Redis.



In this project The voting apps will running in 2 environment, Development and Production. The voting apps in development can be accessed via loadbalancer with name **lks-lb-dev** with 80 and 81 port. while for production it can be accessed via loadbalancer **lks-lb-prod** with opened port 80 and 81.

Deployment LifeCycle



Services should be deployed continuously following the version of the service. All services should be automatically deployed to a docker swarm cluster in ec2 instances for development and ECS for Production, you must use github actions to automate the deployment process. The pipeline will run automatically triggered by code changes in the **dev branch** for development and **prod branch** for production. The public repository used is Github service, has a Docker file in the repository. You can create a docker image using the provided dockerfile, read the

description in each repository to know how to install and create the image. Each image will be uploaded to a private registry with tags and the latest version with a code generation number, for example:

ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/lks-voting-image:vote-{dev/prod}-latest
ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/lks-voting-image:result-{dev/prod}-latest
ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/lks-voting-image:worker-{dev/prod}-latest

You can use the Elastic container registry as a private docker registry. The image you have saved will be deployed to Docker swarm and Elastic Container Service. All these processes must be run in Code Pipeline.

Pre-Requirement

Before you start creating the pipeline you must clone the repository of service into your cloud9, here is the repository of the voting app services:

Voting Service: <https://github.com/sulaimantok/lks-voting-app.git>

Create cloud9 as environment code using **lks-cloud9** as name and **SSH** as connection type. Then, need to fork repository into your github account and clone the source code there. Name the repository as follows **lks-voting-app**. Please read the application installation instruction in README.md. You need to prepare a registry repository with repository names **lks-voting-image** then put it in the registry private repositories.

Add secrets on github repository, The credentials that need to be added to the secrets repository are as follows:

- AWS_ACCESS_KEY
- AWS_SECRET_KEY
- AWS_SESSION_TOKEN
- AWS_SNS_ARN

Reference: <https://docs.github.com/en/actions/security-guides/using-secrets-in-github-actions#creating-secrets-for-a-repository>

Code Pipeline (Github Actions)

Create github actions workflow names **lks-dev-pipeline.yaml**, and **lks-prod-pipeline.yaml**. you can refer to same reference for examples of workflows that can be used, seen in the **.github/workflow/** folder. You must create 3 jobs lks-devpipeline and 2 jobs fot lks-prod-pipeline in the code pipeline with the following job:

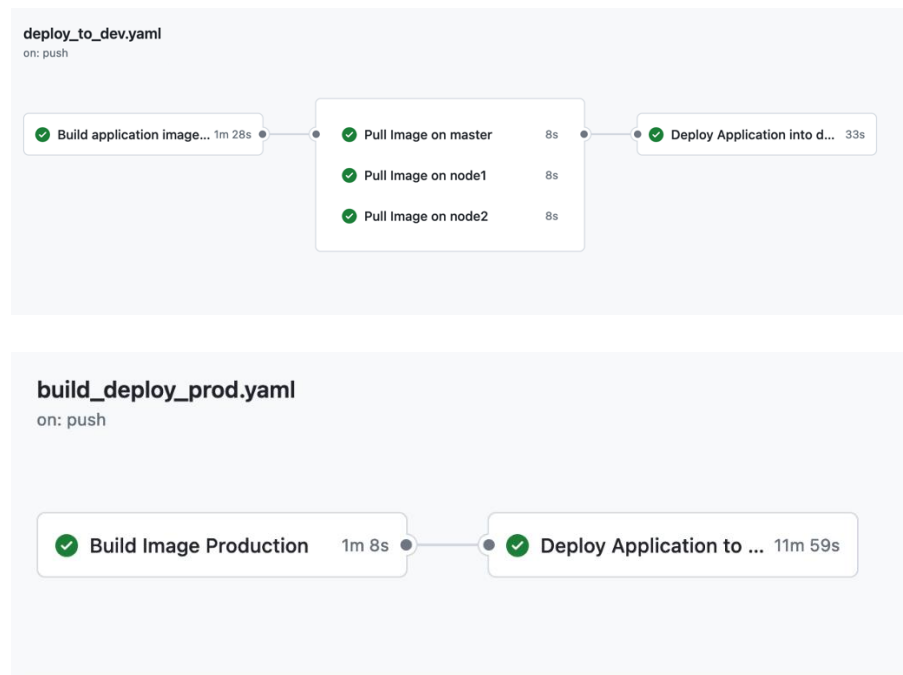
1. lks-dev-pipeline

- **Build** : This job will contain several steps, including update docker version on runner, checkout repository, setup aws credentials, login ECR, build and push image into ECR.
- **Pull Image on master**: This job will contain pull updated image on control plane instance.
- **Pull Image on node1**: This job will contain pull updated image on node1 instance.
- **Pull Image on node2**: This job will contain pull updated image on node2 instance.
- **Deploy** : This job will contain Deploy Application job into Docker swarm cluster.

2. lks-prod-pipeline

- **Build** : This job will contain several steps, including update docker version on runner, checkout repository, setup aws credentials, login ECR, build and push image into ECR.
- **Deploy** : This job will contain update content inside task definition and Deploy Application into ECS Cluster, and send notification if deploy is successful(SNS).

Note: Make sure there are only lks-dev-pipeline and lks-prod-pipeline workflow files in your repository, delete unused workflow files. All jobs must run sequentially, an example of a pipeline image later on GitHub actions as follows.



Service Details

Virtual Private Cloud (VPC)

In this section, you have tasks to create and setup custom VPC named **speaks-vpc** with CIDR 192.168.10.0/24. Ensure the DNS hostnames is enabled and the CIDR must be in three availability zones with the following details:

1. The VPC must contains two public subnets and two private subnets
2. The first public subnet named **speaks-pubsubnet-a** that contains 64 of total hosts with ipv6 enabled. This subnet must be in us-east1-a zone
3. The second public subnet named **speaks-pubsubnet-b** that contains 64 of total hosts after **speaks-pubsubnet-a** prefix. The ipv6 must be enabled and in us-east-b zone.
4. The two private subnets must contain 64 of the total hosts each. Configure in order starting from **speaks-privsubnet-a** that placed on us-east1-a, and **speaks-privsubnet-b** on us-east1-b
5. Configure to make all resources in public subnet able to connect to the internet through an internet gateway named **speaks-igw** and it must forward to 0.0.0.0/0 through **speaks-rtpublic** route table.
6. Configure to make all resources in private subnet able to connect to the internet through NAT Gateway named **speaks-ngw** and it must forward to 0.0.0.0/0 through **speaks-rtpprivate** route table.

Security Groups

Security Group is a feature in Amazon EC2 that has permission to control inbound and inbound traffic that's allowed to reach/leave from the instances that are associated with the security group. In this section, you have to create a few security group with the following details:


1. Named **lks-lb-sg**, inbound traffic 80, and 81 TCP port for 0.0.0.0/0
2. Named **lks-container-sg**, inbound traffic all TCP from **lks-lb-sg**
3. Named **lks-docker-sg**, inbound traffic 22, 5000,5001, 2376, 2377, 7946, 4789 TCP port, and 7946 UDP port.
4. Named **lks-db-sg**, Inbound traffic for PostgreSQL and Redis service from **speaks-vpc**

Docker Swarm (Cluster Development)

Docker Swarm is an orchestration management tool that runs on Docker applications. It helps end-users in creating and deploying a cluster of Docker nodes. Each node of a Docker Swarm is a Docker daemon, and all Docker daemons interact using the Docker API. The Docker swarm in this project will be built and run on AWS EC2. In this section, you have to create a microservice application build on top of Docker swarm to testing application before running in production with the following details:

1. Launch 3 ec2 instances, 1 instance will be control-plane and 2 other instances will be worker. Give the name **lks-master-node** for control plane instance and **lks-node1**, **lks-node2** for 2 other instances.
2. Use **vokey** as a keypair and ensure the instances is accessible from outside
3. Use t2.medium as instance type for all instances
4. Give public ip only on control plane instance
5. Use **lks-docker-sg** for security group

6. Deploy Docker swarm cluster (exec on **lks-master-node**)



```
sudo yum install docker dotnet-sdk-6.0 -y
sudo usermod -aG docker ec2-user
exit
## relogin
docker swarm init

### copy docker swarm join and paste in node1 and node2
```

7. Deploy Docker swarm cluster (exec on **lks-node1** and **lks-node2**)



```
sudo yum install docker dotnet-sdk-6.0 -y
sudo usermod -aG docker ec2-user
exit
## relogin
docker swarm join --token <token> <ip-address>:2377

### command from control plane
```

8. Register all instance as Github self-hosted runner (give label **master** into control plane instance, **node1** for lks-node1 and **node2** for lks-node2)

<https://docs.github.com/en/actions/hosting-your-own-runners/managing-self-hosted-runners/adding-self-hosted-runners>

Amazon Elastic Container Service (ECS)

Elastic container service is used for production. The Voting application should be deployed to ECS via the code pipeline automatically. You must set up an ECS Cluster with name lks-voting-cluster and Fargate as node instances and a VPC with a private subnet (See VPC Service for networking details). You should also create ECS service and task definitions for voting app. You will need the information below to create an ECS cluster, Load balancer, and task definition:

1. Set up vote containers using port 80 for host and port 80 containers.
2. Set up result containers using port 80 for host and port 80 containers.
3. Remove portmapping for worker container
4. Using the base image that has been uploaded in ECR.
5. Use lks-td-vote, lks-td-result, and lks-td-worker for task definition names.
6. Add a container in the task definition with the names lks-vote-container for vote service, lks-result-container for result service and lks-worker-container for worker service.
7. Enable log collection
8. Copy to clipboard json content from task definition, and put it in each service's folder in the repository with name task-definition.json (ex. vote/task-definition.json).

9. Create ECS Cluster with name lks-voting-cluster
10. Create ECS service with the name lks-vote-service, lks-result-service and lks-worker-service with launch type and Application type service.
11. Use lks-lb-prod for the load balancer with the HTTP protocol.
12. Every service use path / for health check.
13. Setup container target group with names lks-tg-vote and lks-tg-result.
14. Create listener 80 to vote container in LB.
15. Create listener 81 to result container in LB.

Microservices Application

In this section, you have tasks to build and deploy microservice application on top of Docker swarm for Development and Amazon ECS for Production. The application is a vote application which connected to the database and Redis. Setup and configure with the following details:

1. Open an AWS Cloud9 and follow the guide to deploy the application here: <https://github.com/sulaimantok/lks-voting-app.git>.
2. Read the README file first and carefully.
3. The Application will be deployed automatically if any changes in github repo. Dev branch for development and prod branch for Production.
4. Application will be accessed via load balancer lks-lb-dev in port 80 and 81 for Development, and lks-lb-prod in port 80, 81 for Production.
5. Ensure all objects are deployed successfully

Amazon RDS

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the AWS Cloud. In this section, you have tasks to setup and configure database in Amazon RDS with the following details:

1. Create a database cluster with using private subnets in two available zones
2. Use postgresQL as a database engine and latest for the version
3. For the template use Free tier.
4. Set lks-rds as a db instance identifier and below for the credentials:
Username: admin
Password: LKSNCC2024
5. Use speaks-vpc and and set public access to no
6. Use lks-db-sg for the security group and turn off for performance insight.
7. Create custom parameter, use custompostgres as name, Postgresql as engine type and latest version postgres for parameter group family.
8. Edit value parameters **rds.force_ssl** into 0 and **password_encryption** into md5.
9. Modify parameter lks-rds to use custompostgres

Amazon ElastiCache

Amazon ElastiCache is a fully managed, in-memory caching service provided by Amazon Web Services (AWS). It allows you to deploy, operate, and scale distributed in-memory caches in the cloud, which can significantly improve the performance and scalability of your applications by reducing the load on backend databases. In this section, you have tasks

to create a file system in Amazon ElasticCache to store cache data from application inside of Amazon ECS with the following details:

1. Create Redis OSS caches with Design your own cache as Deployment option.
2. Cluster cache as Creation method
3. Cluster Mode Disabled
4. Give **lks-redis** for name
5. Disable Multi AZ and enable Auto failover
6. Select latest engine with cache.m7g.large and Number replica 2.
7. Create new subnet group redis-sb as name and select your vpc, and 2 public subnet.
8. Disable automatic backup and disable Auto upgrade minor versions

Amazon Simple Notification Service (SNS)

Amazon Simple Notification Service (SNS) is a fully managed and highly scalable service that manages and facilitates the sending and receiving of messages using a publish/subscribe model to multiple recipients at once. SNS supports several transports, such as HTTP/S, SMS and email. In this task, you must integrate SNS with Github Actions so if any publish into SNS Topic, subscriptions will be notified:

1. Create a topic named lks-vote-topic with standard type
2. Create a subscription using ARN of the topic created before. Set the protocol to Email and fill personallabs4@gmail.com as an endpoint. And confirm to jury if your subs still pending
3. Publish the message below to the topic

Subject: Helo from <Provinsi>!

Message body to send to the endpoint: *New version The Voting App from <Provinsi> is Released !!*