EFS ke 3 ec2

* Ec2 dengan sg webserver basic
* Buat sg baru dengan inbound NFS ke sg webserver
* Buat 1 EFS, manage network nya, sesuaikan ke 3 subnet yg sudah dibuat dalam vpc
* Copy mount url nya
* Tiap ec2 connect
* Sudo –i
* Sudo yum install –y amazon-efs-utils
* Mkdir data
* sudo mount -t efs -o tls fs-04374d383ee89a691:/ data
* ls
* sudah bisa dipakai

Auto scalling with schedule action in specific time

* create AMI image from existed instance
* create Launch template from created AMI , make key pair, assign the existing sg.
* Create ASG from existing launch template, assign the created VPC and 2 subnets of different AZ, choose no load belancer, choose target tracking scalling policy in automatic scalling option. done
* For schedule action , click in created asg, automatic scalling tab, scheduled actions.
* Done

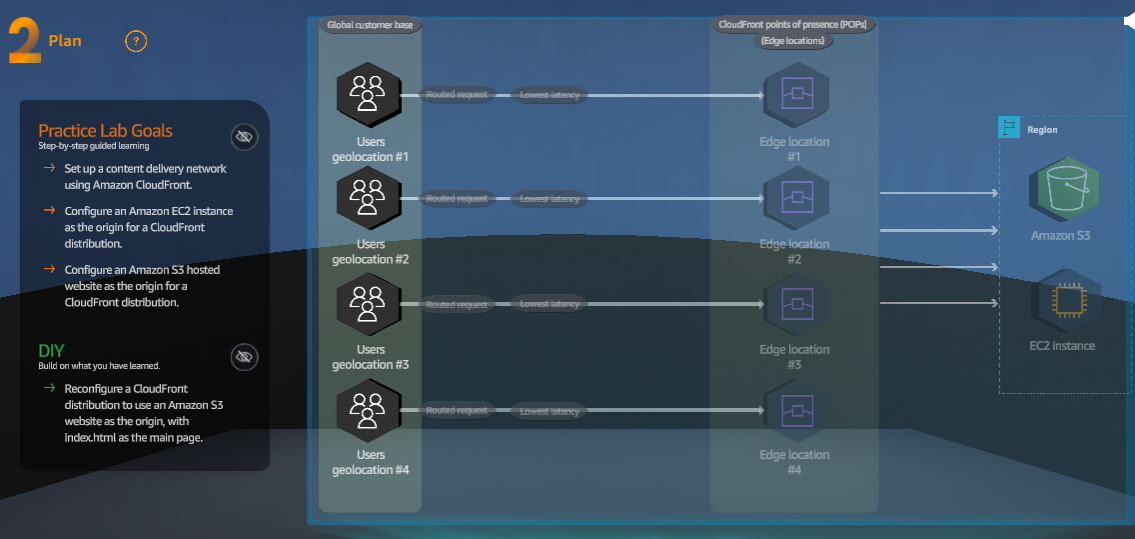
High availability

* Click the created ASG
* Scroll down to load belancer
* Add load belancer, choose application load belancer, choose internet facing, check all the subnets from the vpc, default routing to create a target group,
* Create sgLb with inbound http (anywhere) and outbound http (sgWeb)
* Edit sgWeb with inbound http (sgLb) and outbound http (anywhere) https (anywhere)
* Edit load belancer , security, security group to sgLb
* Copy dns of load belancer and paste to url to see the web
* Click target grups in left , click the target grup, health checks tab, edit , put the healthy check path like /health .
* Click asg, details tab, network, edit, choose only privatesubnet1
* Click instances, choose instance, see if it is in public subnet , terminate it , see there’s gonna be new instance created automatically , see now it is in private subnet. This happens because ASG create it automatically depends on the state of minimum requirement. (you can check the log in activity tab of ASG
* Back to asg, details tab, network, edit, choose all the private subnet 1,2,3
* Back to asg, details tab, group details, edit , set min 1 and max 3

DNS

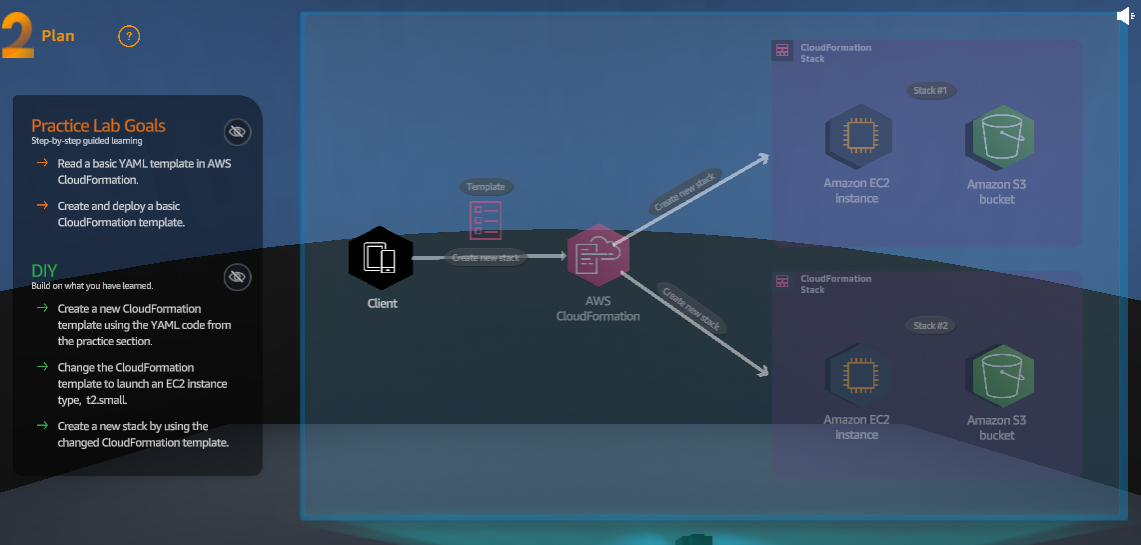
* Ec2, instances, see 2 instances .bastion and internal
* Copy private ip of internal ec2
* Connect to bastion ec2
* Try to ping the copied ip (it connects)
* Now try to ping **internal.news.org** (not connect)
* Click route53,hosted zone,create, type the domain name internal.news.org, type private hosted zone, set the vpc, done
* Create record, record name **thewhitepaper**, type A, value is the copied private IP before, other is default, done
* Try to ping again
* As for the DIY, Now create record, record name **databas**e, type CNAME, value is the newly created A record (**thewhitepaper.internal.news.org**), other is default, done

DNS



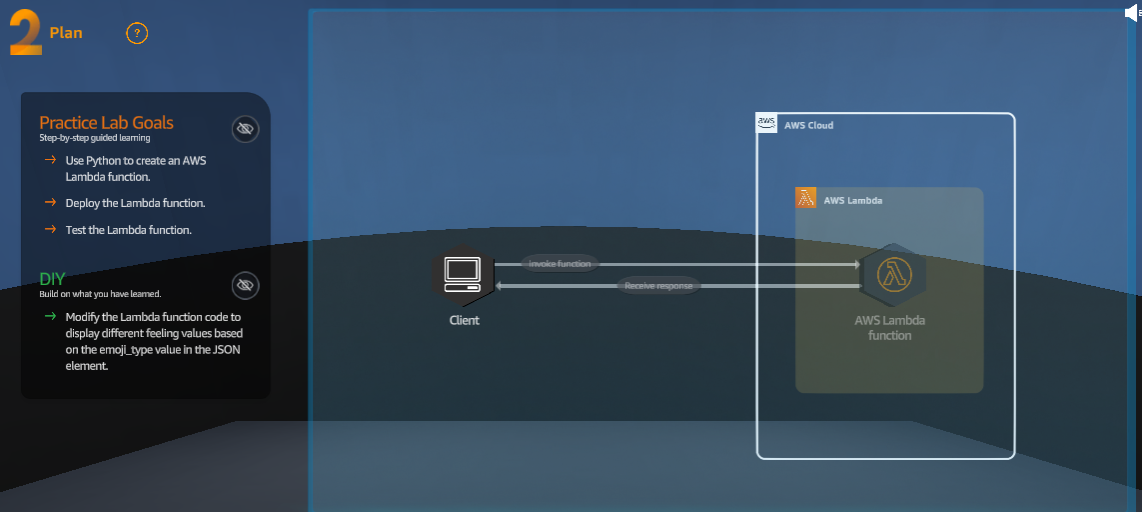
* There is 2 website , 1 from ec2, 1 from s3
* Ec2,instance,copy public ip4 DNS,paste to url , see the web
* Cloudfront,create,paste the copied ip4 dns to origin, set to HTTP only,set cache key and origin reguest to legacy cache, WAF set to do not enable, done
* Copy the distribution domain name, back to distribution tab and wait for status enabled. Paste the domain to url.
  + Note: you can use CNAME as alternate instead of the provided domain from cloudfront
* Go to s3, click the bucket, properties tab, static web hosting, copy the url.
* Do the same as creating cloud front to ec2 before, but this time from s3 url.

Cloud formation



* Download the provided sample\_code.txt
* Cloudformation,create with new resource, set prepare template to build from application composer, click create in application composer, set template, make sure YAML,
* Copy text each line by each line from the downloaded file (the result is at the end of text line).
* You can toggle between template or canvas , yaml or json
* Create template, just next and give name RoboticStack, and next, done
* You can see the log in events or resources tab.
* For more information [AWS::EC2::Instance - AWS CloudFormation (amazon.com)](https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ec2-instance.html#cfn-ec2-instance-instancetype)

Serverless



* Download sample\_code.py
* Lambda, create, author from scratch, choose python 3.11, change default execution role to existing role, create
* Scroll down to code, copy paste the downloaded code, click deploy
* Click test, event sharing to private, template to mobile-backend-echo, change the json to

{

"emoji\_type": 1,

"message": "Hello world!"

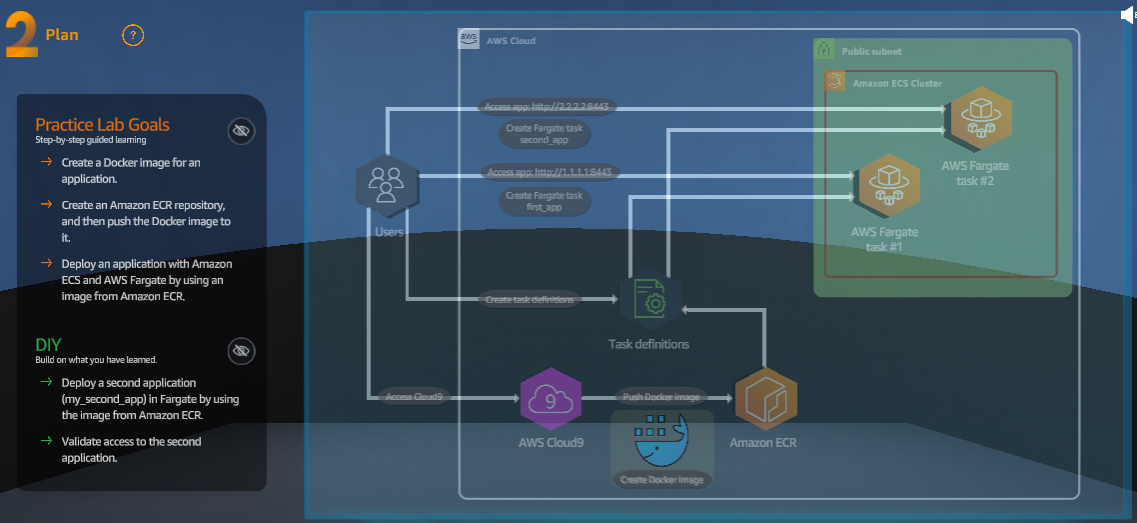
}

* Click save, click test again, see the result test.
* To review the function logs in cloud watch , click monitor tab, view cloudwatch logs, click the log streams
* To change the test just go to test tab, change json, save, test.

More information <https://docs.aws.amazon.com/lambda/latest/dg/python-programming-model.html>

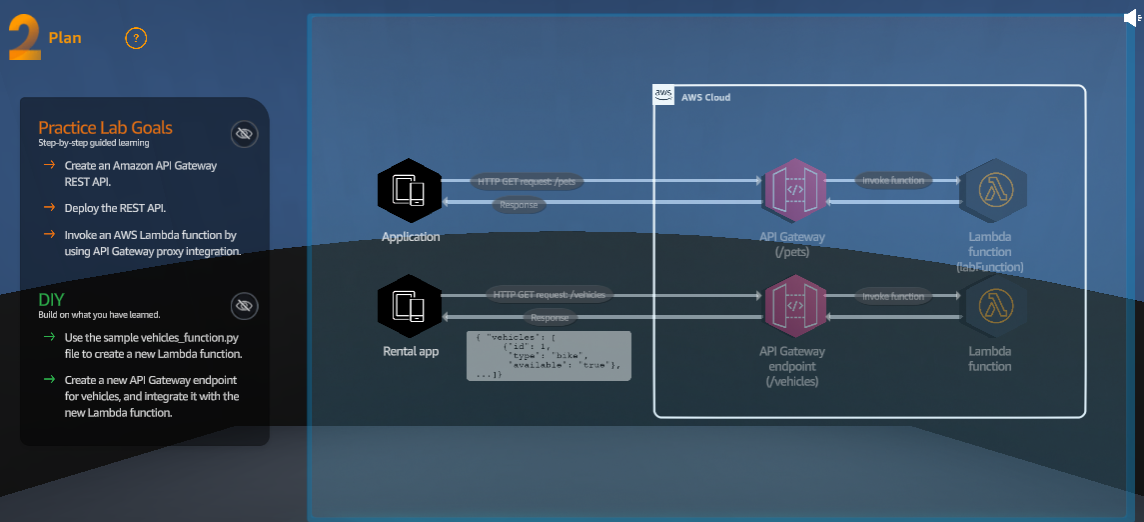
<https://docs.aws.amazon.com/lambda/latest/dg/python-handler.html>

Container



* Download lab\_code.zip
* Cloud9,open
* File,open local files,set the downloaded file.
* Open new terminal ,
* unzip lab\_code.zip
* ./install\_scripts/install\_docker.sh
* region=$(aws configure get region)
* region=${region:-us-east-1}
* repo\_name="my\_app"
* account=$(aws sts get-caller-identity --query Account --output text)
* fullname="${account}.dkr.ecr.${region}.amazonaws.com/${repo\_name}:latest"
* create ECR repo
* aws ecr create-repository --repository-name "${repo\_name}"
* to retrieve auth token
* aws ecr get-login-password --region ${region}|docker login --username AWS --password-stdin ${fullname}
* cd ~/environment/first\_app
* docker build -t ${repo\_name} .
* docker images --filter reference=my\_app
* push to ECR
* docker tag ${repo\_name} ${fullname}
* docker push ${fullname}
* to build and push second app
* cd ~/environment/install\_scripts/
* ./push\_second\_app.sh
* Go to ECR, repositories, my\_app, copy image URI
* Go to VPC, lab\_vpc, copy the vpc id
* Go to SG, select the SG that the VPC ID is the copied vpc id, edit the inbound
* Delete existing inbound , add TCP port 8443 from anywhere ip4, save
* Go to ECS, see cluster was created by Lab
* Task definition tab, create task definition, name first\_app, launch type aws fergate, os linux, task size 5CPU 1gbRAM,
* Scrolldown to container-1, set name my\_app , set image uri to copied uri before, port 8443 TCP, uncheck log collection, create, done
* Deploy, run task, choose existing cluster, Launch type selected, fargate,in deployment set desired task to 1, in network set vpc to labvpc, subnet to publicsubnet1, sg to existing sg,turn on public ip, in task overrides set task and task execution role to the created role, create , done
* Refresh task until its running ,copy the ip4 public, paste to url
* As for DIY , recreate task , but using second app ecr uri , check the website from task ip4 pub

REST ( with api gateway and lambda)



* Download sample\_code(1).py and vehicle\_function.py
* Lambda, create functions of python,check use existing role , set created role (policy = AWSLambdaBasicExecutionRole),create,deploy
* Test,event sharing = private, template = apigateway-aws-proxy,
* For Event JSON, on line 3, type:

"resource": "/pets",

* Create test again by click test,configure test event, set create new event, event name = FindPetById, sharing = private, template = FindAllPets,
* For Event JSON, on line 3, type:

"resource": "/pets/{id}",

* On line 16, type:

"id": 1

* Save
* Api gateway, rest api, build, set new api, set name, endpoint = regional, create, done
* Create resources,resource name = pets, create
* Create method, method type = get, integrate = lambda, check lambda proxy integration, set lambda function to the created lambda function before,create method
* You can test by click test tab,
* In resources click /pets and click create resources, resource name = {id}, create
* Click the {id} and create method again with same get and lambda function before, create
* Test again but with path id = 1
* After test, deploy it, stage = new stage, stage name = lab, deploy
* Copy the invoke url
* Do DIY with lambda and api gateway but different code (vehicle\_function) and /vehicles as url