Deployments

Tuesday, May 6, 2025

12:28 PM

Lambdas:

Most recent 5/6/2025

This is stack named "pr-automated"

* 1. sam build
  2. sam deploy --guided

# DynamoDB

aws cloudformation deploy ^

--template-file dynamo-prreviews.yml ^

--stack-name pr-reviews-dynamo-stack ^

--capabilities CAPABILITY\_NAMED\_IAM ^

--parameter-overrides LambdaExecutionRoleName=pr-automated-PrReviewFunctionRole-j9G7fCetkBSc ^

--no-execute-changeset

If Rollback, need to cleanup

aws cloudformation delete-stack --stack-name pr-reviews-dynamo-stack

If errors, can view events in console or with this cli command:

Aws cloudformation describe-stack-events --stack-name pr-reviews-dynamo-stack

Validate Template

aws cloudformation validate-template --template-body <file://template.yaml>

Describe DDB Table

aws dynamodb describe-table --table-name ChatMemory

# Notifications Service:

Login to ECS:

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 238338230919.dkr.ecr.us-east-1.amazonaws.com

Build

docker build -t notifications-service .

Tag

docker tag notifications-service:latest 238338230919.dkr.ecr.us-east-1.amazonaws.com/notifications-service:latest

Push

docker push 238338230919.dkr.ecr.us-east-1.amazonaws.com/notifications-service:latest

For local testing:

docker run -p 5000:5000 -e ASPNETCORE\_ENVIRONMENT=Development notifications-service

docker run -p 5000:5000 -e ASPNETCORE\_ENVIRONMENT=Development -e AWS\_REGION=us-east-1 notifications-service

docker run -p 5000:5000 -e ASPNETCORE\_ENVIRONMENT=Development -e AWS\_REGION=us-east-1 -e AWS\_PROFILE=default -v /c/Users/issis/.aws:/root/.aws notifications-service

# Overall Infra - VPC, etc. -from a command prompt

aws cloudformation deploy ^

--template-file template.yaml ^

--stack-name pr-notifications-stack ^

--parameter-overrides <file://parameters.json> ^

--capabilities CAPABILITY\_NAMED\_IAM ^

--region us-east-1 ^

--no-execute-changeset

aws cloudformation deploy ^

--template-file template.yaml ^

--stack-name pr-notifications-stack ^

--region us-east-1 ^

--capabilities CAPABILITY\_NAMED\_IAM ^

--no-execute-changeset

aws cloudformation deploy ^

--template-file template.yaml ^

--stack-name pr-notifications-stack ^

--capabilities CAPABILITY\_NAMED\_IAM ^

--parameter-overrides ChatApiTargetGroupArn=arn:aws:elasticloadbalancing:us-east-1:238338230919:targetgroup/chat-api-manual-tg/4832a346e51ddb93

arn:aws:elasticloadbalancing:us-east-1:238338230919:targetgroup/chat-api-manual-tg/4832a346e51ddb93

# Chroma Loader

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 238338230919.dkr.ecr.us-east-1.amazonaws.com

Run from:

C:\Users\issis\source\repos\pull-request-automated-review\chroma\_load

docker build -t chroma-loader .

docker run --rm -v %cd%\chroma\_langchain\_store:/app/chroma\_langchain\_store chroma-loader

Removes all stopped containers and unused images:

docker system prune --all --volumes

Inspect the version of chroma inside the container:

docker run --rm chroma-loader pip show chromadb

# Chroma Server

Running a container with a pre-baked chroma vector store file

Use Chroma Loader to build the chroma db first.

Will have file "chroma.sqlite3" under the chroma\_langchain\_store folder.

From:

C:\Users\issis\source\repos\pull-request-automated-review\chroma\_server

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 238338230919.dkr.ecr.us-east-1.amazonaws.com

docker build -t chroma-server .

For local testing:

docker run --rm -p 8000:8000 chroma-server

docker tag chroma-server:latest 238338230919.dkr.ecr.us-east-1.amazonaws.com/chroma-server:latest

docker push 238338230919.dkr.ecr.us-east-1.amazonaws.com/chroma-server:latest

docker push 238338230919.dkr.ecr.us-east-1.amazonaws.com/notifications-service:latest

A separate repository for chroma-server:

aws ecr create-repository ^

--repository-name chroma-server ^

--region us-east-1

Update the task definition of the service to use the latest:

aws ecs update-service \

--cluster <your-cluster> \

--service <your-service> \

--task-definition PrNotificationsTaskDefinition:<latest\_revision>

aws ecs update-service ^

--cluster pr-notifications-cluster ^

--service pr-notifications-stack-PrNotificationsService-KxCXI1qy4NXU ^

--task-definition pr-notifications-stack-PrNotificationsTaskDefinition-G8PkdY2ft9Qg:1

┌─────────────────────┐

│ notifications-service │

└────────┬────────────┘

│

▼

ALB ─── /notifications/\* ───▶ ECS Task A (notifications)

ALB ─── /chroma/\* ───▶ ECS Task B (chroma-server)

▲

│

chroma\_langchain\_store baked-in or pulled

# Chat API

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 238338230919.dkr.ecr.us-east-1.amazonaws.com

Local testing:

Use the test client for local testing:

* 1. Build: docker build -t chat-ws-app .
  2. Tag: docker tag chat-ws-app:latest 238338230919.dkr.ecr.us-east-1.amazonaws.com/chat-ws-app:latest
  3. Start the container:

    docker run -p 8080:8080 chat-ws-app

* 1. Open pull-request-automated-chat\test\_client\chat\_client.html from a browser.

Use these just when testing locally

docker-compose build

docker-compose build --no-cache

docker-compose up

## Use these when pushing to AWS

docker build -t chat-ws-app .

docker tag chat-ws-app:latest 238338230919.dkr.ecr.us-east-1.amazonaws.com/chat-ws-app:latest

docker push 238338230919.dkr.ecr.us-east-1.amazonaws.com/chat-ws-app:latest

curl <https://api.anthropic.com/v1/messages> ^

-H "x-api-key: $ANTHROPIC\_API\_KEY" ^

-H "anthropic-version: 2023-06-01"

A separate repository for chat-ws-app:

aws ecr create-repository ^

--repository-name chat-ws-app ^

--region us-east-1

✅ Step-by-Step: Manual Deployment of ChatApi

USING POWERSHELL

1. Create Target Group

aws elbv2 create-target-group `

--name chat-api-manual-tg `

--protocol HTTP `

--port 8080 `

--target-type ip `

--vpc-id vpc-0c783d5367931d1a8 `

--health-check-path "/" `

--health-check-protocol HTTP

📌 Save the TargetGroupArn from the output.

{

"TargetGroups": [

{

"TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:238338230919:targetgroup/chat-api-manual-tg/4832a346e51ddb93",

"TargetGroupName": "chat-api-manual-tg",

"Protocol": "HTTP",

"Port": 8080,

"VpcId": "vpc-0c783d5367931d1a8",

"HealthCheckProtocol": "HTTP",

"HealthCheckPort": "traffic-port",

"HealthCheckEnabled": true,

"HealthCheckIntervalSeconds": 30,

"HealthCheckTimeoutSeconds": 5,

"HealthyThresholdCount": 5,

"UnhealthyThresholdCount": 2,

"HealthCheckPath": "/",

"Matcher": {

"HttpCode": "200"

},

"TargetType": "ip",

"ProtocolVersion": "HTTP1",

"IpAddressType": "ipv4"

}

]

}

Perfect — glad that worked in PowerShell! 🎯 That confirms Git Bash’s path rewriting was the blocker.

✅ Next Steps Now That the Target Group Is Created

Now that you have a working Target Group, continue with:

🔹 1. (Optional) Add a Listener Rule

NOTE: execute each of this separately, not multi-line paste.

$conditions = '[{"Field":"path-pattern","Values":["/chat/\*","/ws/\*","/"]}]'

$actions = '[{"Type":"forward","TargetGroupArn":"arn:aws:elasticloadbalancing:us-east-1:238338230919:targetgroup/chat-api-manual-tg/4832a346e51ddb93"}]'

aws elbv2 create-rule --listener-arn arn:aws:elasticloadbalancing:us-east-1:238338230919:listener/app/pr-notifications-alb/f2a975bdfb1e112b/5e95a45de6b23690 --priority 42 --conditions $conditions --actions $actions

🧠 Tip: You can get the listener ARN via:

aws elbv2 describe-listeners --load-balancer-arn arn:aws:elasticloadbalancing:us-east-1:238338230919:loadbalancer/app/pr-notifications-alb/f2a975bdfb1e112b

✅ Guaranteed Fix Using --cli-input-json (Bypasses the problem)

Here’s a workaround that skips JSON parsing issues completely.

🔹 Step 2a: Save this JSON to a file (e.g., create-rule.json):

{  
 "ListenerArn": "arn:aws:elasticloadbalancing:us-east-1:238338230919:listener/app/pr-notifications-alb/f2a975bdfb1e112b/5e95a45de6b23690",  
 "Priority": 42,  
 "Conditions": [  
 {  
 "Field": "path-pattern",  
 "Values": ["/chat/\*", "/ws/\*", "/"]  
 }  
 ],  
 "Actions": [  
 {  
 "Type": "forward",  
 "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:238338230919:targetgroup/chat-api-manual-tg/4832a346e51ddb93"  
 }  
 ]  
}

Save this as create-rule.json in your current directory.

List Rules - to get the ARN

aws elbv2 describe-rules `

--listener-arn arn:aws:elasticloadbalancing:us-east-1:238338230919:listener/app/pr-notifications-alb/f2a975bdfb1e112b/5e95a45de6b23690

arn:aws:elasticloadbalancing:us-east-1:238338230919:listener-rule/app/pr-notifications-alb/f2a975bdfb1e112b/5e95a45de6b23690/ad9855c98fcb8f7f

To delete:

aws elbv2 delete-rule --rule-arn arn:aws:elasticloadbalancing:us-east-1:238338230919:listener-rule/app/pr-notifications-alb/f2a975bdfb1e112b/5e95a45de6b23690/ad9855c98fcb8f7f

🔹 Step 2b: Run this command:

aws elbv2 create-rule --cli-input-json <file://create-rule.json>

✅ This will work reliably — no escaping, no variable weirdness, and it’s 100% AWS CLI–friendly.

Would you like me to generate the equivalent cleanup command (delete-rule) if needed later?

🔹 3. Create the Task Definition

Create a file "register-task-def.json" as follows:

{

  "family": "chat-api-task",

  "networkMode": "awsvpc",

  "requiresCompatibilities": ["FARGATE"],

  "cpu": "512",

  "memory": "1024",

  "executionRoleArn": "arn:aws:iam::238338230919:role/pr-notifications-stack-PrNotificationsTaskExecution-Y4N0OFTDCcoP",

  "containerDefinitions": [

    {

      "name": "chat-ws-app",

      "image": "238338230919.dkr.ecr.us-east-1.amazonaws.com/chat-ws-app",

      "portMappings": [

        {

          "containerPort": 8080,

          "protocol": "tcp"

        }

      ],

      "logConfiguration": {

        "logDriver": "awslogs",

        "options": {

          "awslogs-group": "/ecs/chat-ws-app",

          "awslogs-region": "us-east-1",

          "awslogs-stream-prefix": "ecs"

        }

      }

    }

  ]

}

Run this command:

aws ecs register-task-definition --cli-input-json <file://register-task-def.json>

🔹 2. Register ECS Fargate Service to Test It

Create a file "create-ecs-service.json".

{

  "cluster": "pr-notifications-cluster",

  "serviceName": "chat-api-test",

  "taskDefinition": "chat-api-task",

  "launchType": "FARGATE",

  "desiredCount": 1,

  "networkConfiguration": {

    "awsvpcConfiguration": {

      "subnets": ["subnet-0b2933f32469d1d20", "subnet-0842140ad3ff98009"],

      "securityGroups": ["sg-0125f1582b0801c21"],

      "assignPublicIp": "ENABLED"

    }

  },

  "loadBalancers": [

    {

      "targetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:238338230919:targetgroup/chat-api-manual-tg/4832a346e51ddb93",

      "containerName": "chat-ws-app",

      "containerPort": 8080

    }

  ]

}

Replace:

* 1. Subnet IDs
  2. Security Group ID
  3. containerName and containerPort from your task def

Run this command:

aws ecs create-service --cli-input-json <file://create-ecs-service.json>

🔹 3. Monitor Target Health

aws elbv2 describe-target-health `  
 --target-group-arn <your-tg-arn>

Look for TargetHealth -> State == healthy.

If everything turns green, we’ll know CloudFormation is failing due to timing or race conditions — and we can either re-import or simplify the CF template.

Would you like help auto-fetching the right listener ARN and target group now?

🔁 After Testing

Once this works:

* 1. You can confirm CloudFormation’s failure is due to resource creation timing or reuse
  2. Optionally import working resources into a new CF stack using aws cloudformation import (if needed)

Let me know if you’d like help building exact commands for your resource IDs — just share your VPC ID, subnet IDs, security group ID, and listener ARN.

From <<https://chatgpt.com/c/682a410f-bf30-8010-ae7e-50e6beeac719>>

# pr\_git

aws cloudformation deploy \

--template-file git-provider-ecs.yml \

--stack-name git-provider-service-stack \

--capabilities CAPABILITY\_IAM \

--parameter-overrides <file://parameters-git-provider.json>

docker build -t git-provider-api .