Cloud Architectures

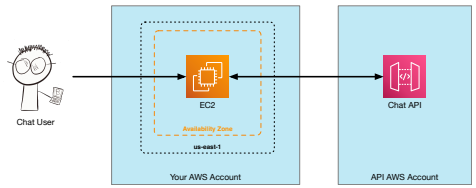
Cloud Architectures

Reading and Understanding Architecture Diagrams

- Architecture diagrams are a visual overview of the major pieces involved in an application
- Can be as sparse or detailed as you need
- Usually tailored to the audience
  - Developers want to see a more detailed diagram
  - Executives want to see a higher level diagram

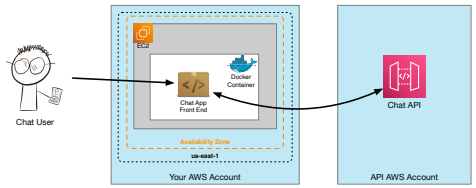
Cloud Architectures

High Level Chat App Diagram



Cloud Architectures

More Detailed Chat App Diagram



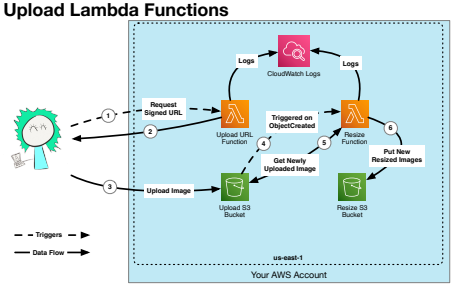
Cloud Architectures

Image Upload Lambda Functions

- How about a diagram for Homework 7?
- What were the pieces we had?
  - 2 S3 Buckets
  - 2 Lambda Functions
  - A user
  - An Image File
  - Resized Images

Cloud Architectures

Image Upload Lambda Functions



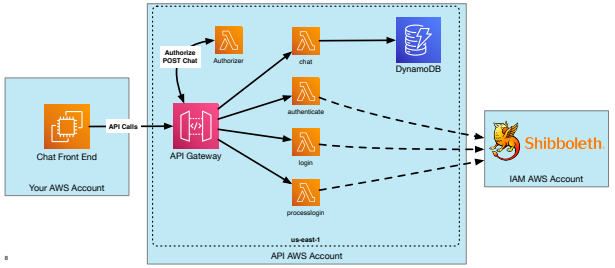
Cloud Architectures

Chat API Back End

- What does the Architecture Diagram look like for the Chat API back end?
- Major building blocks
  - API Gateway
  - Lambda
  - DynamoDB

Cloud Architectures

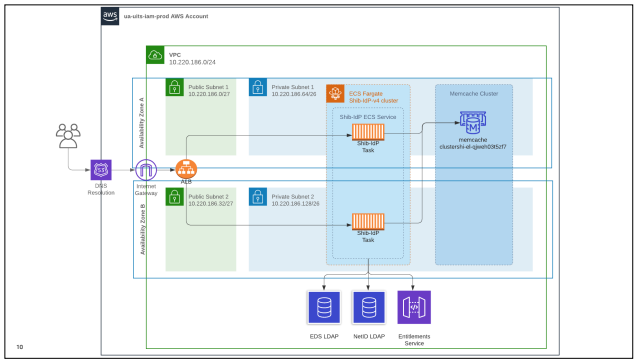
Chat API Back End



Cloud Architectures

Shibboleth / WebAuth Architecture Diagram

- How about a more robust service, like the main campus WebAuth Identity Provider?
- Services Involved
  - Application Load Balancer
  - Elastice Container Service (ECS)
  - AWS ElastiCache (memcache)
  - VPC Subnets and Availability Zones



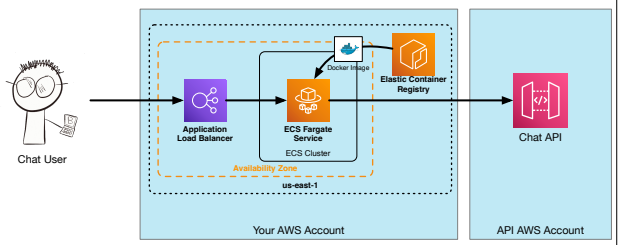
Cloud Architectures

Chat App v2

- How could we improve our initial Chat App architecture?
- Elastic Container Service
  - No EC2 instance to manage
  - Use Elastic Container Registry to store and retrieve Docker Images
- Application Load Balancer
  - Scale to multiple back end containers
  - Allow for easier SSL/TLS termination

Cloud Architectures

Chat App v2 Architecture



Application Load Balancer & Elastic Container Service

Application Load Balancer

- Public HTTP Endpoint
- Distribute incoming requests to multiple back-end processes
- HTTPS / SSL termination
- PaaS - AWS worries about patching and scaling
- Can perform some basic routing based on paths or protocols
  - Incoming HTTP → HTTPS
  - Static files to S3, dynamic requests to code

Elastic Container Service

- Runs Docker containers
- Stores Docker images
- Automatically maps load balancer to container ports
- Can be configured to scale the number of back end containers
- Can run on a managed set of EC2 instances, or completely serverless with Fargate

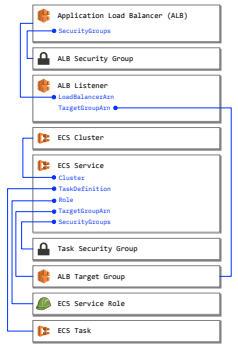
Application Load Balancer & Elastic Container Service

Demo

ALB & ECS

Automation?

- Many resources needed
- Possible by hand, but many chances to make mistakes
- Infrastructure as Code to the rescue
- CloudFormation Template



CloudFormation

ALB + ECS Template

- Parameters
  - Inputs to the template
  - By abstracting out parameters, a single template can be deployed multiple times and in multiple accounts

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "iam-admin"

Metadata:
  cfn-lint:
    config:
      regions:
        - us-west-2
      ignore_checks:
        - I3042

Parameters:
  DockerImage:
    Type: String

  LabRoleARN:
    Type: String

  VpcId:
    Type: AWS::EC2::VPC::Id

  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>

  AcmCertificateArn:
    Type: String


Resources:
  LoadBalancer:
```

CloudFormation

ALB + ECS Template

- All these parameters will be unique to each account
- You will need to look up these values for your account


```
AWSTemplateFormatVersion: "2010-09-09"
Description: "iam-admin"

Metadata:
  cfn-lint:
    config:
      regions:
        - us-west-2
      ignore_checks:
        - I3042

Parameters:
  DockerImage:
    Type: String

  LabRoleARN:
    Type: String

  VpcId:
    Type: AWS::EC2::VPC::Id

  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>

  AcmCertificateArn:
    Type: String


Resources:
  LoadBalancer:
```

CloudFormation

Load Balancer

- To create an Application Load Balancer, we need to know what security group to attach to it, and what subnets it belongs to.
- SubnetIds comes from our input Parameters
- Security Group is defined in this template and referenced here
- Other properties are hard-coded (type, scheme, etc)


```
    Type: String

  LabRoleARN:
    Type: String

  VpcId:
    Type: AWS::EC2::VPC::Id

  SubnetIds:
    Type: List<AWS::EC2::Subnet::Id>

  AcmCertificateArn:
    Type: String


Resources:
  LoadBalancer:
    Type: AWS::ElasticLoadBalancingV2::LoadBalancer
    Properties:
      SecurityGroups:
        - !Ref LoadBalancerSecurityGroup
      Scheme: internet-facing
      Subnets: !Ref SubnetIds
      IpAddressType: ipv4
      Type: application

  LoadBalancerHttpListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: "redirect"
          RedirectConfig:
            Protocol: "HTTPS"
            Port: "443"
            Host: "#{host}"
```

CloudFormation

Listeners

- Since this is an HTTP endpoint, we need to specify which ports to listen on
- Port 80 listener redirects all traffic to port 443
- Port 443 listener sends requests to the Target Group
  - Linked to a Certificate from input Parameters


```
      IpAddressType: ipv4
      Type: application

  LoadBalancerHttpListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: "redirect"
          RedirectConfig:
            Protocol: "HTTPS"
            Port: "443"
            Host: "#{host}"
            Path: "/#{path}"
            Query: "#{query}"
            StatusCode: "HTTP_301"
      LoadBalancerArn: !Ref LoadBalancer
      Port: 80
      Protocol: "HTTP"

  LoadBalancerHttpsListener:
    Type: AWS::ElasticLoadBalancingV2::Listener
    Properties:
      DefaultActions:
        - Type: forward
          TargetGroupArn: !Ref LoadBalancerTargetGroup
      LoadBalancerArn: !Ref LoadBalancer
      Port: 443
      Certificates:
        - CertificateArn: !Ref AcmCertificateArn
      Protocol: HTTPS

  LoadBalancerTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      TargetType: ip
```

CloudFormation

Target Group

- Target Group links the ALB and listener to an ECS service
- Needs to be attached to the same VPC that our ALB subnets are in
- Healthcheck is defined


```
      LoadBalancerArn: !Ref LoadBalancer
      Port: 443
      Certificates:
        - CertificateArn: !Ref AcmCertificateArn
      Protocol: HTTPS

  LoadBalancerTargetGroup:
    Type: AWS::ElasticLoadBalancingV2::TargetGroup
    Properties:
      TargetType: ip
      TargetGroupAttributes:
        - Key: deregistration_delay.timeout_seconds
          Value: 30
      HealthCheckEnabled: true
      HealthCheckIntervalSeconds: 60
      HealthCheckPath: "/"
      HealthCheckPort: "80"
      HealthCheckProtocol: HTTP
      HealthCheckTimeoutSeconds: 5
      HealthyThresholdCount: 3
      Matcher:
        HttpCode: 200-299
      Port: 80
      Protocol: HTTP
      VpcId: !Ref VpcId
    DependsOn:
      - LoadBalancer

  LoadBalancerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: !Sub "${AWS::StackName} external security gro
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: "tcp"
```

CloudFormation

ALB Security Group

- Here's the security group referenced by the Application Load Balancer
- Needs to allow incoming traffic on ports 80 and 443


```
      Port: 80
      Protocol: HTTP
      VpcId: !Ref VpcId
    DependsOn:
      - LoadBalancer

  LoadBalancerSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: !Sub "${AWS::StackName} external security gro
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: "tcp"
          CidrIp: "0.0.0.0/0"
          ToPort: 80
          FromPort: 80
        - IpProtocol: "tcp"
          CidrIp: "0.0.0.0/0"
          ToPort: 443
          FromPort: 443
      SecurityGroupEgress:
        - IpProtocol: "-1"
          CidrIp: "0.0.0.0/0"

  TaskSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: !Sub "${AWS::StackName} internal security gro
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: "tcp"
          SourceSecurityGroupId: !Ref LoadBalancerSecurityGroup
          ToPort: 80
          FromPort: 80
      SecurityGroupEgress:
```

CloudFormation

ECS Task Group

- The security group that surrounds the Container Task only allows traffic from objects in the Load Balancer security group
- Principle of least privilege: Only allow in traffic you absolutely need to. Nothing besides the ALB needs to send traffic to the containers


```
          ToPort: 80
          FromPort: 80
        - IpProtocol: "tcp"
          CidrIp: "0.0.0.0/0"
          ToPort: 443
          FromPort: 443
      SecurityGroupEgress:
        - IpProtocol: "-1"
          CidrIp: "0.0.0.0/0"

  TaskSecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      GroupDescription: !Sub "${AWS::StackName} internal security gro
      VpcId: !Ref VpcId
      SecurityGroupIngress:
        - IpProtocol: "tcp"
          SourceSecurityGroupId: !Ref LoadBalancerSecurityGroup
          ToPort: 80
          FromPort: 80
      SecurityGroupEgress:
        - IpProtocol: "-1"
          CidrIp: "0.0.0.0/0"

  Cluster:
    Type: AWS::ECS::Cluster
    Properties:
      ClusterName: !Ref AWS::StackName

  Service:
    Type: AWS::ECS::Service
    DependsOn:
      - LoadBalancerHttpListener
    Properties:
      Cluster: !Ref Cluster
```

CloudFormation
ECS Cluster

- The ECS Cluster itself is a very simple resource. It's really just a named container for other things to be attached to


```
TaskSecurityGroup:
  Type: AWS::EC2::SecurityGroup
  Properties:
    GroupDescription: !Sub "${AWS::StackName} internal security gro
    VpcId: !Ref VpcId
    SecurityGroupIngress:
      - IpProtocol: "tcp"
        SourceSecurityGroupId: !Ref LoadBalancerSecurityGroup
        ToPort: 80
        FromPort: 80
    SecurityGroupEgress:
      - IpProtocol: "-1"
        CidrIp: "0.0.0.0/0"

Cluster:
  Type: AWS::ECS::Cluster
  Properties:
    ClusterName: !Ref AWS::StackName

Service:
  Type: AWS::ECS::Service
  DependsOn:
    - LoadBalancerHttpsListener
  Properties:
    Cluster: !Ref Cluster
    ServiceName: !Ref AWS::StackName
    DeploymentConfiguration:
      MaximumPercent: 200
      MinimumHealthyPercent: 50
    DesiredCount: 1
    HealthCheckGracePeriodSeconds: 30
    LaunchType: FARGATE
    LoadBalancers:
      - ContainerName: !Sub "${AWS::StackName}-task"
```

CloudFormation
ECS Service

- An ECS Service defines an always-running set of container tasks
- Connects the ALB target group to actual containers
- FARGATE is the AWS serverless model for containers


```
Cluster:
  Type: AWS::ECS::Cluster
  Properties:
    ClusterName: !Ref AWS::StackName

Service:
  Type: AWS::ECS::Service
  DependsOn:
    - LoadBalancerHttpListener
  Properties:
    Cluster: !Ref Cluster
    ServiceName: !Ref AWS::StackName
    DeploymentConfiguration:
      MaximumPercent: 200
      MinimumHealthyPercent: 50
    DesiredCount: 1
    HealthCheckGracePeriodSeconds: 30
    LaunchType: FARGATE
    LoadBalancers:
      - ContainerName: !Sub "${AWS::StackName}-task"
        ContainerPort: 80
        TargetGroupArn: !Ref LoadBalancerTargetGroup
    NetworkConfiguration:
      AwsvpcConfiguration:
        Subnets: !Ref SubnetIds
        SecurityGroups:
          - !Ref TaskSecurityGroup
        AssignPublicIp: ENABLED
    TaskDefinition: !Ref TaskDefinition

TaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Cpu: 512
```

CloudFormation
ECS Task Definition

- The Task Definition defines all the properties for a given container, or set of containers
- Analogous to the docker run command
  - What image to run
  - What container port
  - Where do logs go
- ECS Service automatically maps host ports to the container port


```
        AssignPublicIp: ENABLED
    TaskDefinition: !Ref TaskDefinition

TaskDefinition:
  Type: AWS::ECS::TaskDefinition
  Properties:
    Cpu: 512
    Memory: 1024
    NetworkMode: awsvpc
    TaskRoleArn: !Ref LabRoleARN
    ExecutionRoleArn: !Ref LabRoleARN
    RequiresCompatibilities:
      - FARGATE
    ContainerDefinitions:
      - Name: !Sub "${AWS::StackName}-task"
        Image: !Ref DockerImage
        PortMappings:
          - ContainerPort: 80
        LogConfiguration:
          LogDriver: awslogs
          Options:
            awslogs-group: !Ref TaskLogGroup
            awslogs-region: !Ref AWS::Region
            awslogs-stream-prefix: "app"


TaskLogGroup:
  Type: AWS::Logs::LogGroup
  DeletionPolicy: Delete
  UpdateReplacePolicy: Delete
  Properties:
    LogGroupName: !Sub "${AWS::StackName}-logs"
    RetentionInDays: 7
```

CloudFormation
Log Group

- Lastly we define a Log Group where the ECS task logs will be delivered
- By explicitly creating it, we can specify the retention policy
- If we let it be automatically created, logs stay around forever!

```
      ContainerDefinitions:
        - Name: !Sub "${AWS::StackName}-task"
          Image: !Ref DockerImage
          PortMappings:
            - ContainerPort: 80
          LogConfiguration:
            LogDriver: awslogs
            Options:
              awslogs-group: !Ref TaskLogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: "app"


  TaskLogGroup:
    Type: AWS::Logs::LogGroup
    DeletionPolicy: Delete
    UpdateReplacePolicy: Delete
    Properties:
      LogGroupName: !Sub "${AWS::StackName}-logs"
      RetentionInDays: 7

Outputs:
  LoadBalancerDNSName:
    Value: !GetAtt LoadBalancer.DNSName
    Export:
      Name: !Sub "${AWS::StackName}-alb-dns-name"
```


CloudFormation
Outputs

- We want to get the DNS name of the load balancer, so we can point a friendly DNS entry at it
  - CNAME

```
              awslogs-group: !Ref TaskLogGroup
              awslogs-region: !Ref AWS::Region
              awslogs-stream-prefix: "app"


  TaskLogGroup:
    Type: AWS::Logs::LogGroup
    DeletionPolicy: Delete
    UpdateReplacePolicy: Delete
    Properties:
      LogGroupName: !Sub "${AWS::StackName}-logs"
      RetentionInDays: 7

Outputs:
  LoadBalancerDNSName:
    Value: !GetAtt LoadBalancer.DNSName
    Export:
      Name: !Sub "${AWS::StackName}-alb-dns-name"
```