How do you handle the chicken-and-egg problem with dependencies? (Demonstrates architectural thinking.)

Or

Design a CloudFormation architecture for a microservices application with shared resources (VPC, ALB) and service-specific resources.

What is the Chicken-and-Egg Problem in AWS CloudFormation?

It's when **two stacks depend on each other**, but neither can be created **before** the other — causing **deployment failures**.

🔁 For example:

- Stack A needs something from Stack B.
- Stack B also needs something from Stack A.
- But CloudFormation can't create both at the same time — so it fails.

Real Example (Microservices and Load Balancer)

Imagine you're building a system like Amazon:

You have:

- A **shared Load Balancer (ALB)** like a "traffic controller".
- Multiple **services** like UserService, OrderService.

Each service wants:

- To **connect to ALB** (so users can reach it via paths like /users, /orders)
- But the ALB also needs to **know about those services' target groups** (to route traffic correctly).

Now here's the twist:

- ALB Stack **can't wait** for the services.
- Services **can't work** unless ALB exists.

🎯 That's the chicken-and-egg problem!

✅ **How to Solve It**

🔷 **1. Use a Layered Stack Design**

Layer	What it Includes	Why it's Important
Layer 1	VPC, Subnets, NAT Gateways	Basic Networking (Foundation)
Layer 2	ALB, Listener, Default Target Groups	Shared resources (Reusable by all apps)
Layer 3	Microservices (ECS, Rules)	Your actual business logic (Apps)

🔷 **2. Add Dummy (Default) Target Groups**

- Create **empty placeholders** in the ALB stack.
- Add **default listeners** (which return 404 errors).
- Later, microservices add their **real routing rules**.

🔄 This avoids tight dependency between ALB and services.

🔷 **3. Use CloudFormation Exports and Imports**

Let stacks **share information** without hardcoding.

In ALB Stack:

yaml

```
Outputs:
  AlbArn:
    Value: !Ref ApplicationLoadBalancer
    Export:
      Name: SharedStack-AlbArn
```

In Service Stack (UserService):

yaml

CopyEdit

```
Parameters:
  AlbArn:
    Type: String
    Default: !ImportValue SharedStack-AlbArn
```

This means:

- ALB shares its info.
- Services import it when they need it.

🔷 **4. Define a Deployment Order**

Create a small script to deploy stacks **in order**:

bash

```
#!/bin/bash
```

aws cloudformation deploy --template-file network.yaml --stack-name shared-network
aws cloudformation deploy --template-file alb.yaml --stack-name shared-alb
aws cloudformation deploy --template-file user-service.yaml --stack-name user-service

---

Best Practices

1. **Break into layers** – Helps avoid circular dependencies.
2. **Export/Import values** – No need to hardcode ARNs.
3. **Default listeners** – So ALB is ready even before services exist.
4. **Use Parameter Store** – For configs like DB URLs.
5. **Health checks** – So only healthy ECS tasks receive traffic.
6. **Separate CI/CD pipelines** – One for shared infra, one per microservice.
7. **Tag resources** – Easier billing, tracking, and debugging.

Final Takeaway:
You **don't need to create everything at once**.
Instead, break your infrastructure into **smaller parts**, let them **share data using exports/imports**, and use **dummy placeholders** to avoid dependency loops.

1. What is the main purpose of separating CloudFormation stacks into Foundation, Shared Services, and Service Layers?
A. To reduce the template file size
B. To follow the principle of least privilege
C. To promote modular, independent deployments and avoid circular dependencies
D. To reduce the cost of CloudFormation stacks
Answer: C

2. What problem arises when ALB target groups are created by microservice stacks but also referenced in the shared ALB stack?
A. Permission denied error
B. Circular dependency error
C. Stack policy violation
D. Stack deletion protection error
Answer: B