Cloud Computing Case Study: Council of Plumbers Cloud Modernization Project

Background

The **Council of Plumbers (CoP)** is the national licensing and certification body for plumbers. With **30,000 active members**, the Council manages member profiles, certifications, and monthly training exams via an online portal.

Currently, CoP operates a **monolithic application** hosted on aging infrastructure. Each month, thousands of plumbers log in to upload exam results and certifications. This causes:

- Severe **performance degradation** during peak periods.
- **Server crashes** due to traffic spikes.
- **Data inconsistencies** caused by retries or failed transactions.

Problem Summary

- The system **cannot handle concurrent connections** efficiently.
- The monolithic design **limits scalability and maintainability**.
- There is **no auto-scaling or dynamic resource management**.
- The client requires **zero downtime** during the migration.

Project Objective

Modernize the Council of Plumbers' system by:

1. **Decomposing the monolith into microservices and APIs**.
2. **Containerizing services with Docker and Kubernetes (or ECS/Fargate)**.
3. **Implementing Infrastructure as Code (IaC)** for scalable cloud infrastructure.
4. **Ensuring zero downtime during the migration**.

Phased Deliverables

Phase 1: Assessment & Design

Deliverables:

- Document current architecture and workflows.
- Identify core services to break out (e.g., User Login, Exam Upload, Certification Management).
- Map out data dependencies and shared state.

Guidelines:

- Use **API-first design**.
- Start with **read-heavy services**.
- Plan for **blue/green or canary deployments**.

Phase 2: Infrastructure Setup

Deliverables:

- Set up cloud infrastructure (AWS/GCP/Azure or local Kubernetes).
- Deploy **Load Balancer and API Gateway**.
- Prepare **Container Registry**.
- Define **auto-scaling policies** using Infrastructure as Code.

Guidelines:

- Use **Terraform** or **CloudFormation** for IaC.
- Set up **Kubernetes HPA** or **Auto Scaling Groups**.
- Integrate **monitoring and logging**.

Phase 3: API & Microservices Development

Deliverables:

- Break out at least **two core services**.
- **Containerize** using Docker.
- Deploy services to Kubernetes or ECS.

Guidelines:

- Use **RESTful APIs** with OpenAPI documentation.
- Implement **health checks**.
- Use **retry logic and circuit breakers**.

Phase 4: Database Modernization

Deliverables:

- Decide on **database strategy** (shared DB vs service DB).
- Set up **read replicas** to handle read spikes.
- Plan for **incremental data migration** if needed.

Guidelines:

- Use **change data capture (CDC)** for updates.
- Backup data before changes.

Phase 5: Deployment & Cutover

Deliverables:

- Use **blue/green or canary deployments**.
- Implement **rollback plans**.
- Gradually route traffic from monolith to new services.

Guidelines:

- Start with **10% traffic routing**.
- Monitor with **RUM, logs, and dashboards**.
- Complete cutover after stability checks.

Infrastructure as Code Examples

AWS Auto Scaling (Terraform Example)

```
resource "aws_launch_template" "plumbers_app" {
  name_prefix   = "plumbers-app-"
  image_id      = "ami-xxxxxxx"
  instance_type = "t3.medium"

  user_data = base64encode(file("startup-script.sh"))
}

resource "aws_autoscaling_group" "plumbers_asg" {
  desired_capacity     = 2
  max_size             = 5
  min_size             = 1
  vpc_zone_identifier  = ["subnet-xxxx", "subnet-yyyy"]

  launch_template {
    id      = aws_launch_template.plumbers_app.id
    version = "$Latest"
  }
}

resource "aws_autoscaling_policy" "scale_out" {
  name                   = "scale-out"
  autoscaling_group_name = aws_autoscaling_group.plumbers_asg.name
  scaling_adjustment     = 1
  adjustment_type        = "ChangeInCapacity"
  cooldown               = 300
}
```

Kubernetes Horizontal Pod Autoscaler (HPA)

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: plumbers-api-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: plumbers-api
  minReplicas: 2
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 60
```

Key Learning Outcomes

- Understand **real-world cloud migration constraints**.
- Apply **Infrastructure as Code (IaC)**.
- Develop **microservices and APIs**.
- Use **container orchestration**.
- Implement **scaling policies and monitoring**.
- Manage **zero downtime deployments**.

Bonus Challenges

- Implement **CI/CD pipelines**.
- Add **API security (OAuth2, JWT)**.

• Use **serverless functions** for lightweight background tasks.