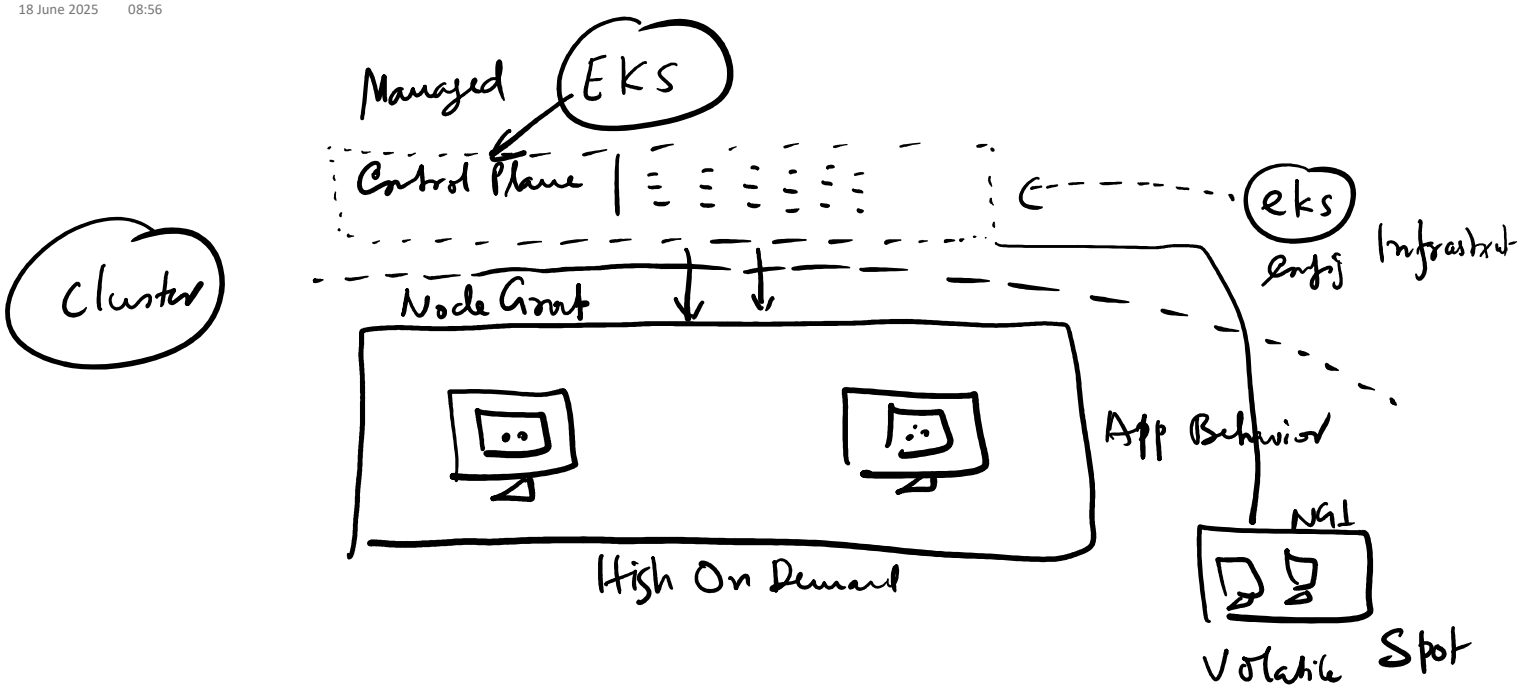
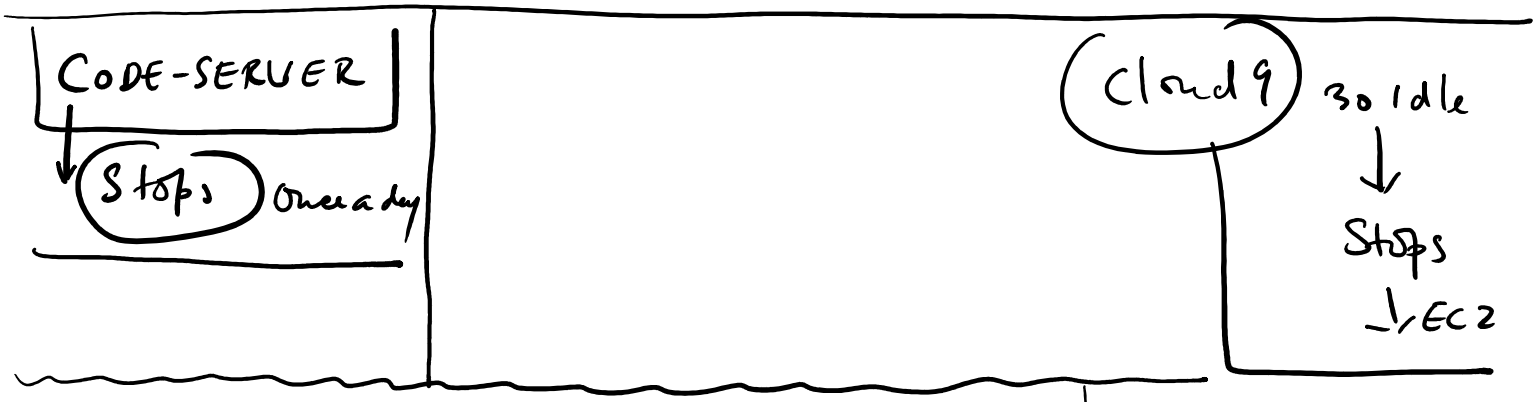
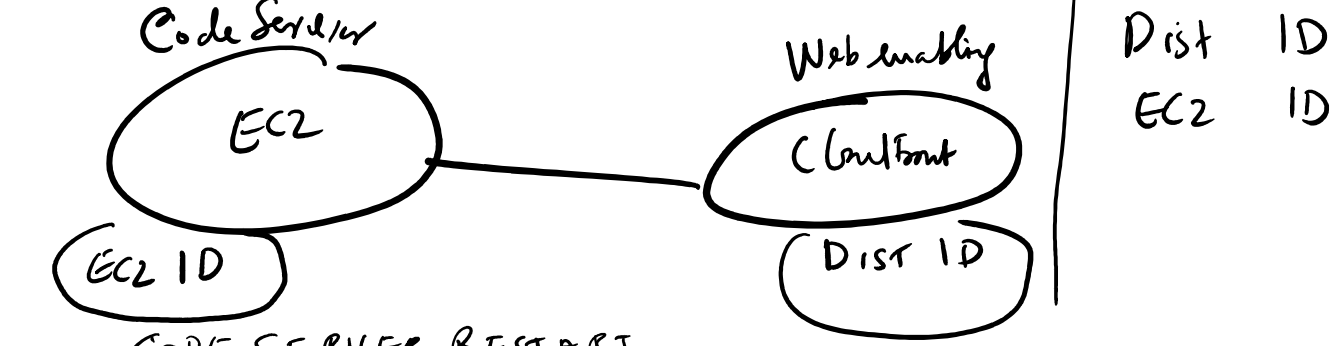
Day02

18 June 2025 08:56







CODE-SERVER RESTART

(1) EDIT Restart-IDE.sh

→ EC2-ID : Get it from your IDE EC2
→ DIST-ID : Get it from your CloudFront Dist

(2) Cloudshell – in the same region

(2) CloudShell – in the same region

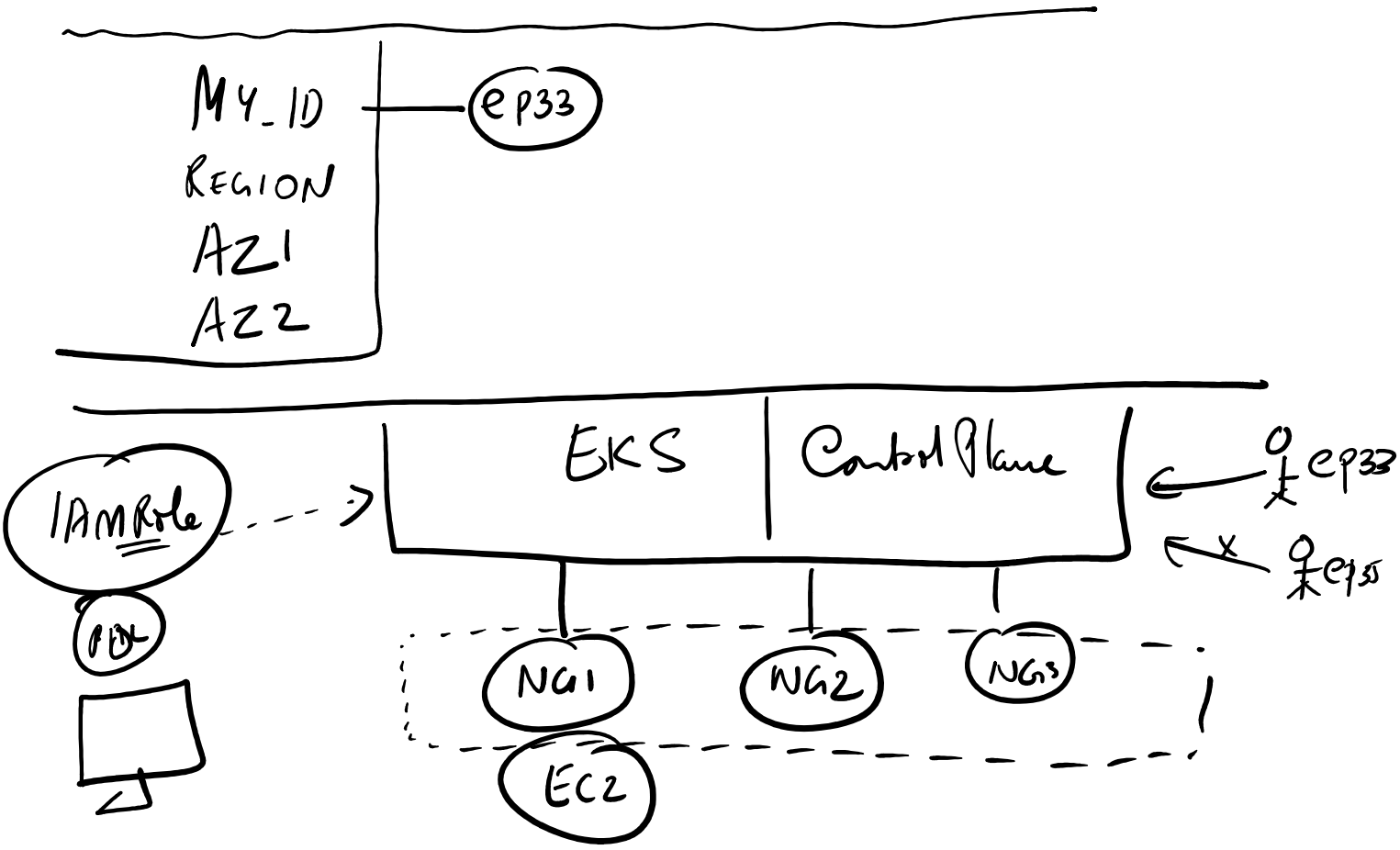
→ nano Restart-IDE.sh

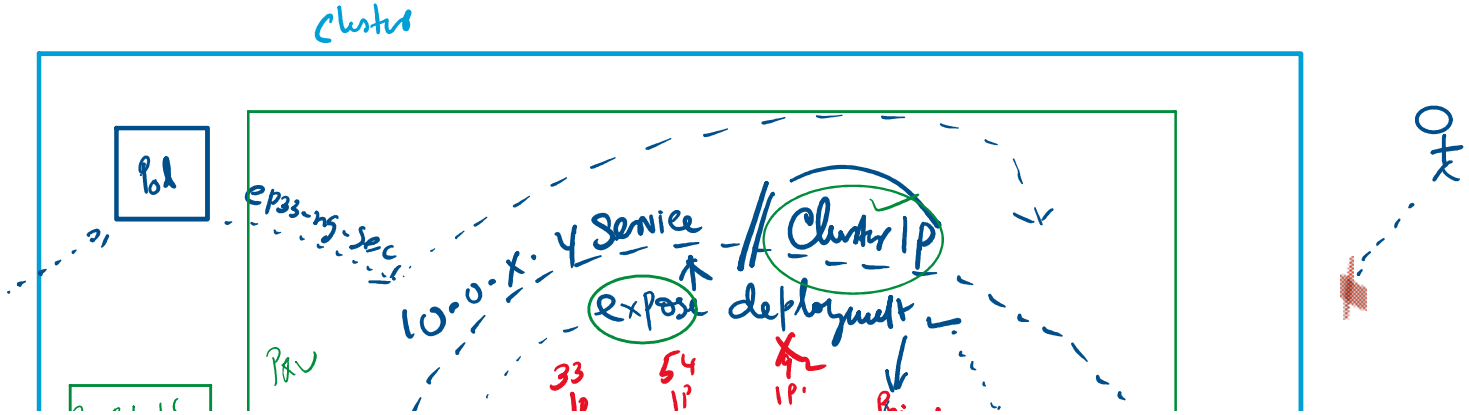
=) Copy paste the shell script

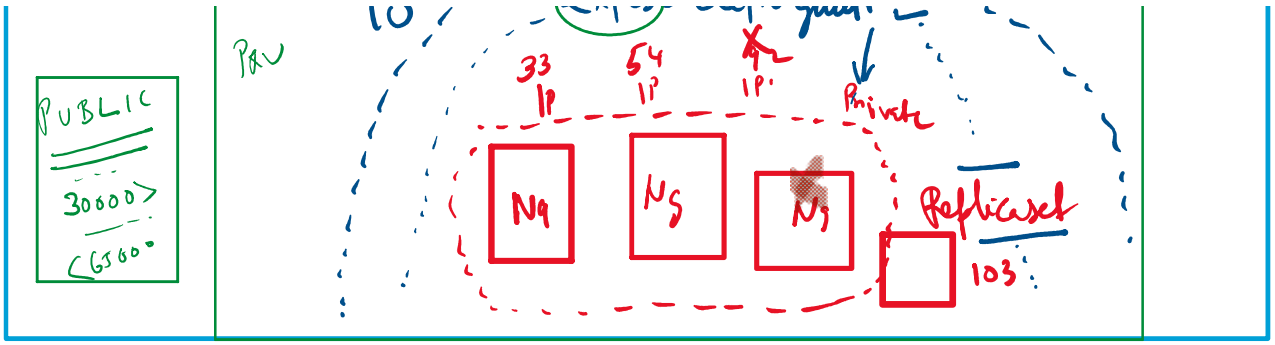
CH + S + CH + X

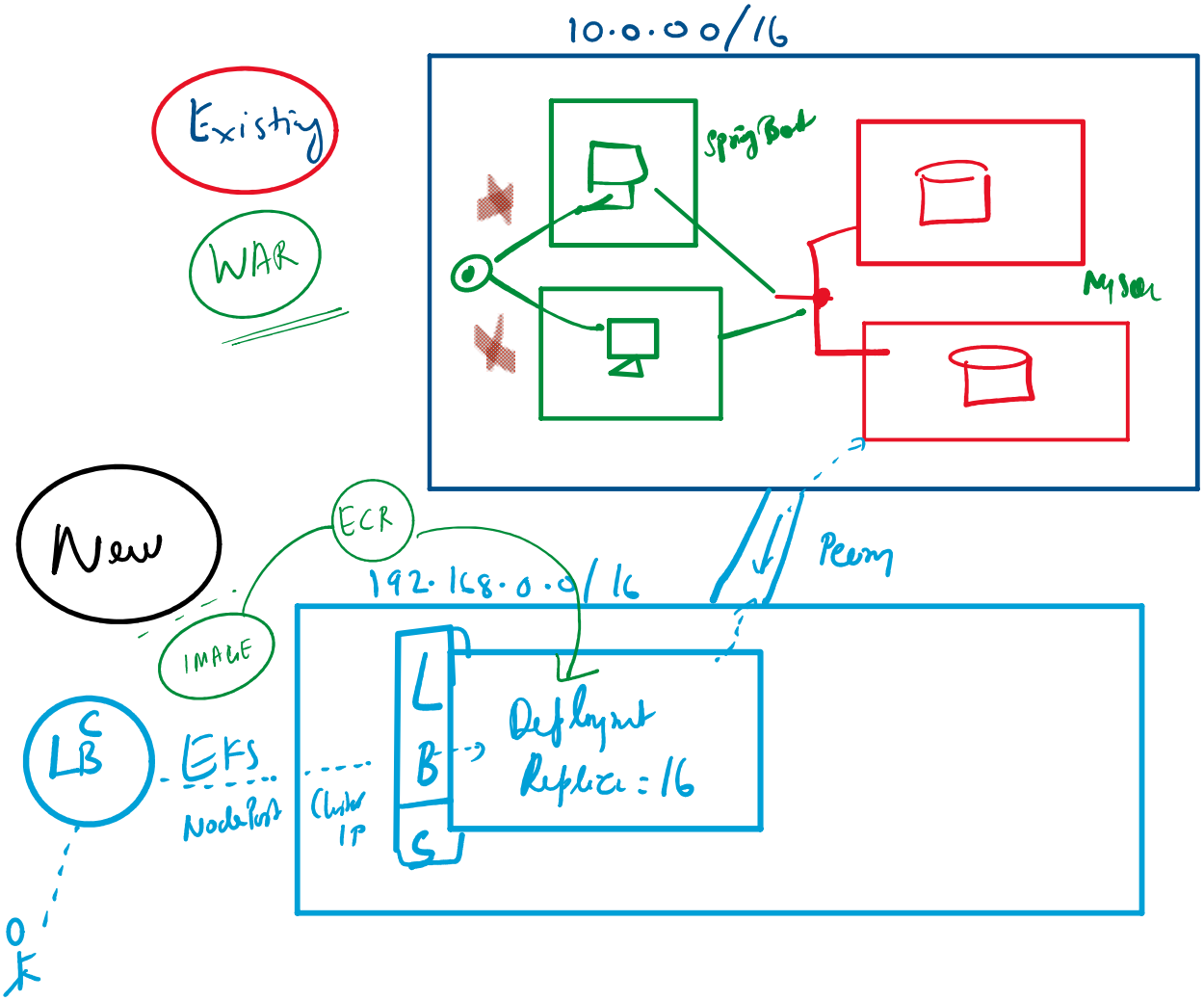
(3) chmod +x Restart-IDE.sh

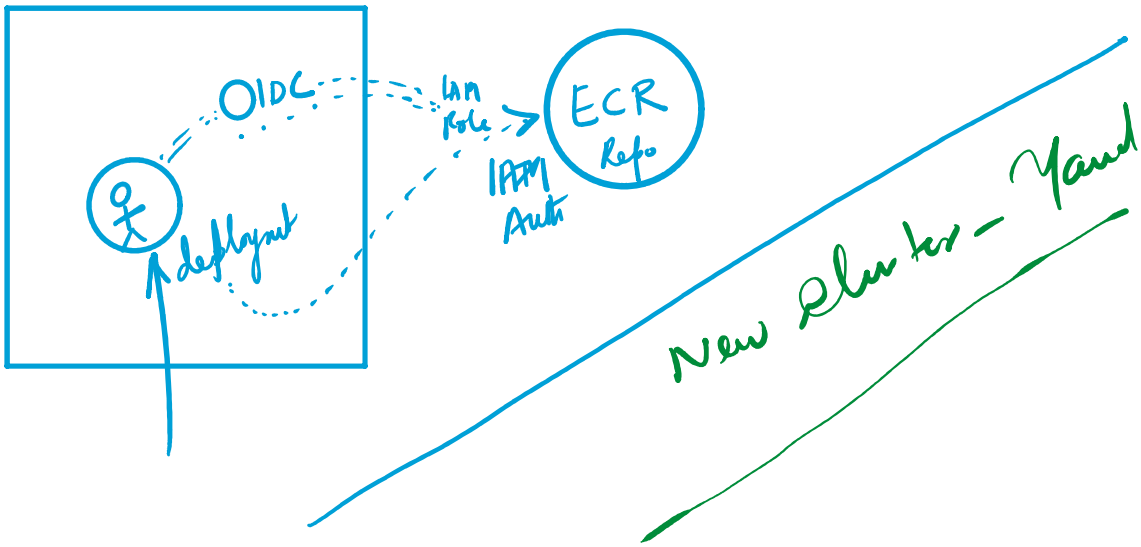
(4) ./Restart-IDE.sh






PUBLIC
30000>
<6500
33 IP
54 IP
IP
Private
Replicaset
103


10.0.00/16
Existing
WAR
SpringBoot
MySql
New
ECR
IMAGE
192.168.0.0/16
Peering
LB
EKS
NodePort
Cluster IP
LB
Deployment
Replica : 16



Technical Explanation: EKS Microservices Lab

1. Microservices Architecture in this Context

Microservices architecture involves breaking an application into discrete, independently deployable services that communicate over well-defined APIs. Each microservice runs independently, often within its own containerized environment. In this lab context, we deploy a microservices demo application (sock-shop) using Helm and Kubernetes. Each microservice (e.g., front-end, catalogue, orders) is packaged into a separate container, facilitating independent development, deployment, and scaling. Kubernetes (EKS specifically) orchestrates these microservices, ensuring resilience, high availability, load balancing, and automated service discovery. The combination of Kubernetes and Helm simplifies deployment complexity by defining infrastructure and application components declaratively.

2. Ingress and Load Balancer Behavior

Ingress

An Ingress resource in Kubernetes manages external HTTP/HTTPS access to services inside the cluster. Ingress provides routing based on paths and hostnames, facilitating easy and scalable exposure of services externally. In this lab, we use the Ingress-NGINX controller as the implementation to handle ingress resources.

Load Balancers (Classic / NLB / ALB)

- **Classic ELB:**
  - Layer 4 and 7 load balancing
  - Simple, legacy option, less granular control.
- **Network Load Balancer (NLB):**
  - Operates at Layer 4 (Transport Layer - TCP/UDP).
  - Extremely performant, handles millions of requests per second.
  - Ideal for high-performance, low-latency workloads.
- **Application Load Balancer (ALB):**
  - Operates at Layer 7 (Application Layer - HTTP/HTTPS).
  - Offers advanced routing rules based on request paths and headers.
  - Ideal for microservices architectures with multiple routes and complex rules.

Choosing Load Balancer:

- For HTTP/HTTPS with advanced routing: Use **ALB**.
- For high-performance TCP/UDP applications: Use **NLB**.
- For simple use cases (legacy apps): Use **Classic ELB**.

- For HTTP/HTTPS with advanced routing: Use **ALB**.
- For high-performance TCP/UDP applications: Use **NLB**.
- For simple use cases (legacy apps): Use **Classic ELB**.

3. Advantages and Disadvantages of Using Helm

Advantages:

- **Declarative Management:** Infrastructure defined in code.
- **Versioning:** Easy rollback and version management.
- **Reusability:** Helm charts can be shared and reused.
- **Templating:** Flexible and dynamic configuration for different environments.
- **Simplicity:** Single command deploys complex applications.

Disadvantages:

- **Complexity:** Additional abstraction layer may obscure direct Kubernetes configurations.
- **Learning Curve:** Users must understand Helm syntax and chart structure.
- **Dependency Management:** Improper handling of chart dependencies can lead to confusion or errors.

4. Charts vs. Templates

- **Helm Charts:** A Helm chart is a packaged application or service, containing multiple Kubernetes resource templates and configuration metadata (Chart.yaml).
- **Templates:** Individual YAML files with Go-templated placeholders, included within a Helm chart, dynamically generating Kubernetes resources based on provided values.

A **Chart** is the high-level packaging of your app, whereas **Templates** are building blocks used by Helm to render final Kubernetes manifests.

5. Chart Dependencies and Lock Files

- **Chart Dependencies:** Defined in Chart.yaml or previously in requirements.yaml, they specify other Helm charts required by the main chart. Managed by helm dependency commands.
- **Lock Files (Chart.lock or requirements.lock):** Generated by Helm, ensuring that the exact version of dependencies is used consistently across deployments, improving reliability and reproducibility.

Managing dependencies properly ensures consistency and stability across environments, preventing unexpected behavior due to version discrepancies.

6. Deep Dive into ingress.yaml

Typical ingress.yaml structure:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: sock-shop
  namespace: sock-shop
  annotations:
    kubernetes.io/ingress.class: nginx
spec:
  ingressClassName: nginx
  rules:
    - host: example.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: front-end
```

```
      port:
        number: 80
```

Other options and annotations:

- **TLS:** Secure ingress with TLS by specifying certificates.
- **Path Types:** Prefix, Exact, or ImplementationSpecific to determine path matching behavior.
- **Annotations:** Enable advanced behaviors like URL rewrite, sticky sessions, CORS, rate limiting, or authentication.

7. Additional Considerations Not Discussed

- **Horizontal Pod Autoscaler (HPA):** Automates pod scaling based on metrics.
- **Resource Quotas and Limits:** Ensures fair resource usage among namespaces.
- **Security Best Practices:** Pod Security Standards, RBAC configuration.
- **Observability:** Integration with Prometheus/Grafana for monitoring.
- **Backup and Restore:** Using tools like Velero for disaster recovery.
- **GitOps Approach:** Integrating GitOps tools (ArgoCD, Flux) for fully automated deployments.

Incorporating these practices enhances operational maturity, reliability, and maintainability of your EKS-based microservices architecture.

1. Classic ELB (Legacy, Not Recommended for New Deployments)

By default, creating a Service of type LoadBalancer (without special annotations) usually provisions a **Classic ELB** on older clusters, but most new EKS clusters default to NLB or ALB based on annotation.

```
apiVersion: v1
kind: Service
metadata:
  name: my-classic-lb
spec:
  type: LoadBalancer
  ports:
    - port: 80
      targetPort: 8080
  selector:
    app: my-app
```

default = CLB

Note: Classic ELB support is being phased out in favor of NLB and ALB.

2. Network Load Balancer (NLB)

To explicitly provision an **NLB**, use the following annotation in your Service YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: my-nlb
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
spec:
  type: LoadBalancer
  ports:
```

```
    - port: 80
      targetPort: 8080
  selector:
    app: my-app
```

3. Application Load Balancer (ALB)
ALB is not provisioned directly via Service.
Instead, you use the **AWS Load Balancer Controller** and define an **Ingress** resource with specific annotations and class.

- **Install AWS Load Balancer Controller first.**
- Then, create an Ingress like:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-alb-ingress
  annotations:
    kubernetes.io/ingress.class: alb
    alb.ingress.kubernetes.io/scheme: internet-facing
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: my-app
                port:
                  number: 80
```

This instructs the AWS Load Balancer Controller to provision an **ALB** for you.

Quick Reference Table

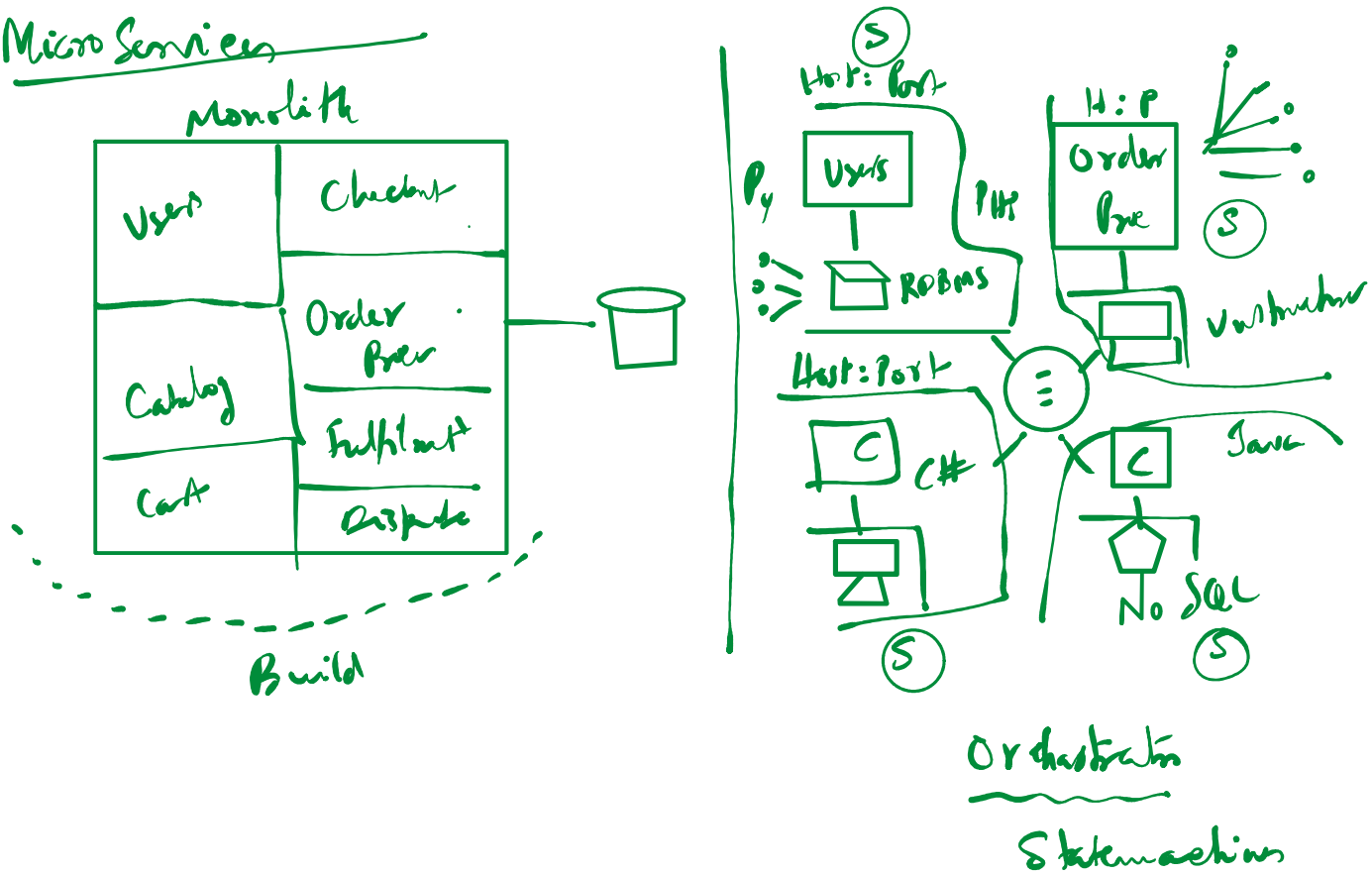
Type	Resource	How to Select
Classic ELB	Service	Default (older clusters), or legacy annotation
NLB	Service	Annotation: service.beta.kubernetes.io/aws-load-balancer-type: "nlb"
ALB	Ingress	Use AWS Load Balancer Controller + ingress.class: alb

Summary

- **For NLB:** Add the annotation on a Service manifest.
- **For ALB:** Use Ingress with AWS Load Balancer Controller and correct class/annotations.
- **For Classic ELB:** Default or legacy; new workloads should prefer NLB/ALB.

Pro Tip:
For production, prefer **NLB for TCP/UDP workloads** and **ALB for advanced HTTP/HTTPS routing**.
If you want example manifests for each, let me know!

1 Micro Services



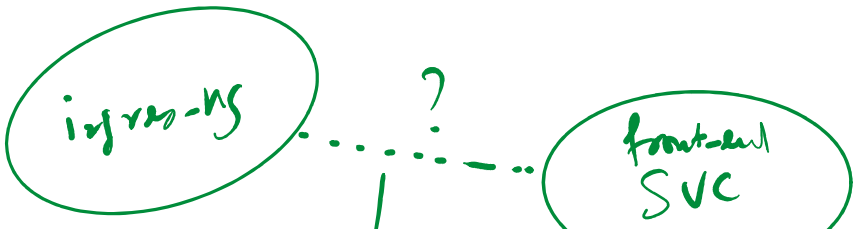
Orchestration

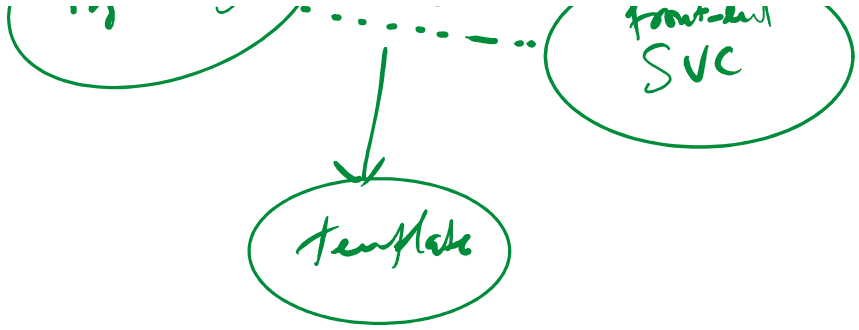
Statemachine

Diff @clusterwide ingress/nginx

Vs

loadbalancer Service





In our EKS case - EKS is sitting in Tier2 NAT subnet. You cannot access that subnet from outside VPC. Thus you need to create LoadBalancer and point it to EKS pods.

In our EKS we used to allow app to provide scaling rules for ASGs associated with nodegroups. But that leads to instability of EKS nodes because EKS won't know about ASG rules and ASG won't know about EKS. There were incidents (not in Prod, thankfully), where nodes were constantly going up and down because of ASG scaling rules being defined.