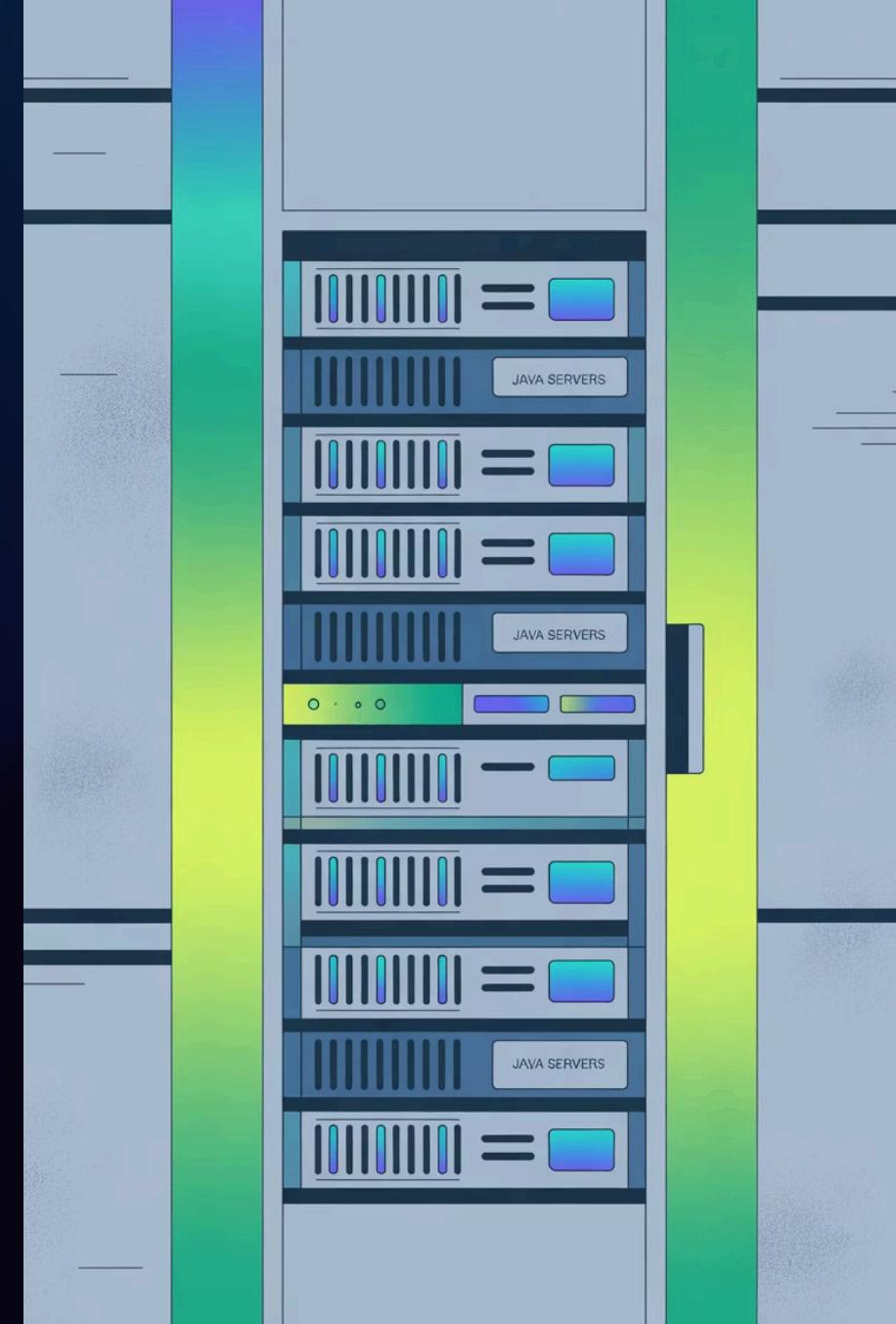


# **Implementando Clustering em Open Liberty**

## **Alta Disponibilidade e Escalabilidade em Ambientes On-premise e VM**

Uma abordagem técnica e objetiva para administradores de sistemas e desenvolvedores Java



# Contexto: Web Server em Liberty

## Servidor de Aplicações Completo

O Open Liberty não é apenas um web server, mas um servidor de aplicações Java completo, compatível com Jakarta EE e MicroProfile. Ele hospeda:

- Servlets, JSP, APIs REST
- Aplicações monolíticas e microsserviços
- Integração nativa com HTTP/HTTPS

## Diferente do Apache HTTP Server

Enquanto o Apache atua como camada de entrega (estático/dinâmica via módulos), o Liberty **processa e executa aplicações empresariais Java**, eliminando a necessidade de um application server adicional.

## Pontos Fortes do Open Liberty

- Inicialização **muito rápida** (segundos)
- Baixo consumo de memória/CPU
- Modularidade (features ativadas sob demanda via server.xml)
- Fácil integração com ambientes on-premise ou cloud

# O que é Clustering em Liberty (Sem Kubernetes)

No modo tradicional, clustering no Liberty é feito com [Collectives](#):

## Controller

Servidor central que gerencia configuração e monitora o cluster.

## Members

Nós que rodam as aplicações e se reportam ao controller.

## Load Balancer

Distribui requisições entre os members (Apache, NGINX, IHS).

## Benefícios

- Alta disponibilidade (HA)
- Escalabilidade horizontal
- Gerenciamento centralizado
- Replicação de sessão (para aplicações stateful)

## Quando usar sem Kubernetes

- Ambientes on-premise ou VM-based
- Quando não há orquestrador cloud-native
- Cenários legados que precisam HA mas não querem migrar toda a infraestrutura

Controlador

Membro 2

# Arquitetura do Cluster Tradicional



## Controller

Usa a feature collectiveController-1.0

## Members

Usam collectiveMember-1.0 e clusterMember-1.0

## Segurança

Comunicação segura via SSL/TLS (ssl-1.0)

- Sessões replicadas com sessionCache-1.0 + provedor (JCache, Infinispan, Hazelcast).
- Load Balancer geralmente ativo-ativo para evitar ponto único de falha.

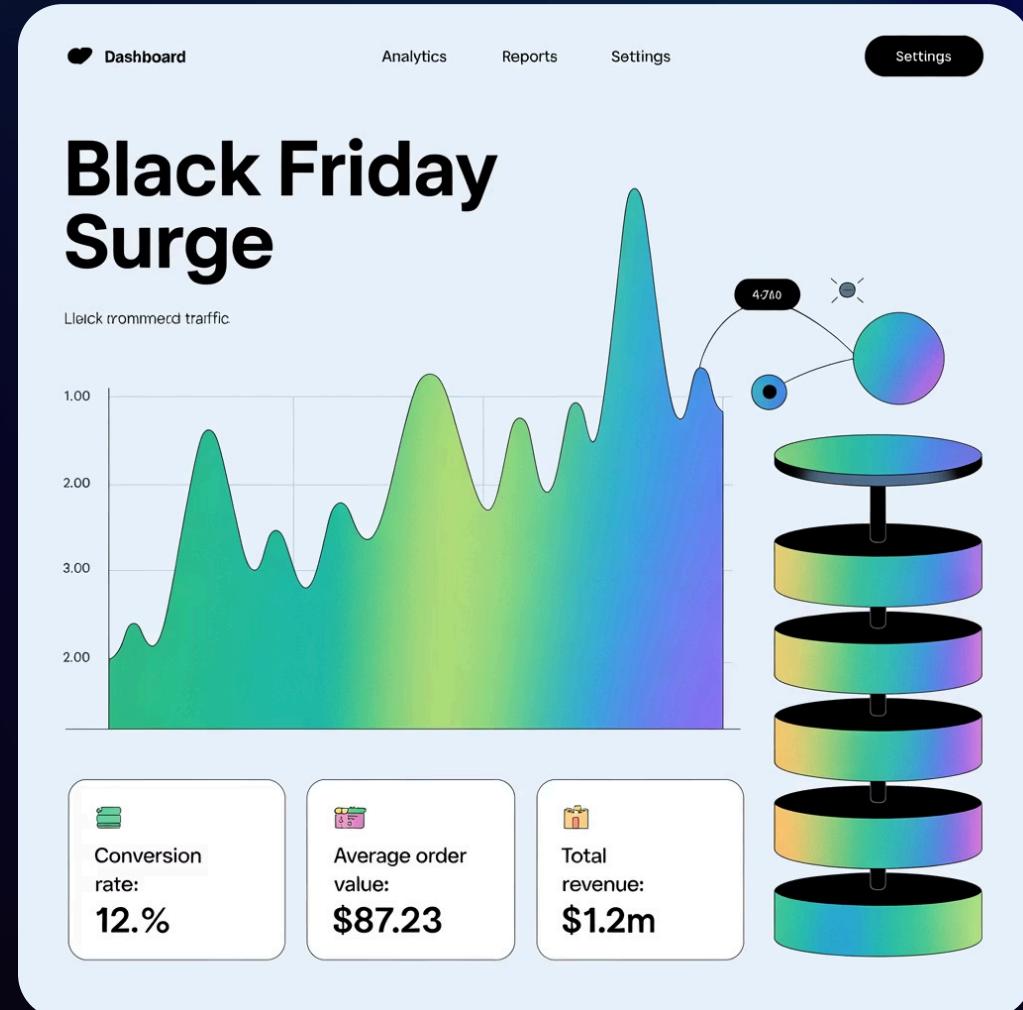
# Importância do Clustering

## Alta Disponibilidade

- **Failover automático**: se um member falha, outro assume a carga
- Minimiza downtime (atende SLAs altos como 99.9% ou 99.99%)

## Escalabilidade

- Adição/remoção de nós conforme demanda
- Melhor uso de recursos físicos



## Exemplo Real

E-commerce em datas de pico (Black Friday) escala para 6 nós; no restante do ano, opera com 3.

# Cálculo do Número de Nós (Sem K8s)

## Fórmula básica

$$Nós = (RPS_{esperado}/RPS_{por\ n\'o}) + redund\ ^{\^a}ncia$$

### RPS esperado

Obtido com testes de carga (JMeter, Locust)

### RPS por nó

Medido em ambiente de homologação

### Redundância

Mínimo N+1 (um nó extra)

#### Dica importante:

- Sempre usar pelo menos 3 nós para quorum e HA
- Ajustar após monitoramento real (Prometheus ou JVisualVM)

# Itens de Atenção (Sem Kubernetes)



## Segurança

- Habilitar ssl-1.0 para criptografia
- Usar certificados válidos e renovar antes do vencimento
- Restringir portas (9080 HTTP, 9443 HTTPS) no firewall



## Rede

- Latência entre nós < 50ms
- Evitar SPoF no load balancer (usar dois LB em ativo-ativo)



## Performance

- Ajustar heap (-Xms = -Xmx)
- Pool de conexões de BD ajustado ao tamanho do cluster
- Evitar replicação excessiva de sessão (impacta latência)

# Configuração Passo a Passo - Cluster Tradicional

## Pré-requisitos:

- Java 8+
- Liberty instalado (java -jar wlp-install.jar --acceptLicense)
- Usuário admin (wasadmin)

## 1. Criar Controller

```
./server create wlpController  
./collective create wlpController \  
--keystorePassword=senha  
  
# Adicionar ao server.xml:  
# collectiveController-1.0  
# ssl-1.0  
  
./server start wlpController
```

## 2. Criar Member

```
./server create wlpMember1  
./collective join wlpMember1 \  
--host=IP_CONTROLLER --port=9443 \  
--user=wasadmin --password=senha \  
--keystorePassword=senha  
  
# Adicionar ao server.xml:  
# collectiveMember-1.0  
# clusterMember-1.0  
  
./server start wlpMember1
```

## Sessões e HA

- Habilitar sessionCache-1.0
- Configurar provedor

# Best Practices para Clusters Sem Kubernetes

## Automação

Scripts de deploy e configuração para múltiplos nós. Use ferramentas como Ansible ou Jenkins para automatizar processos repetitivos.

## Rolling Updates

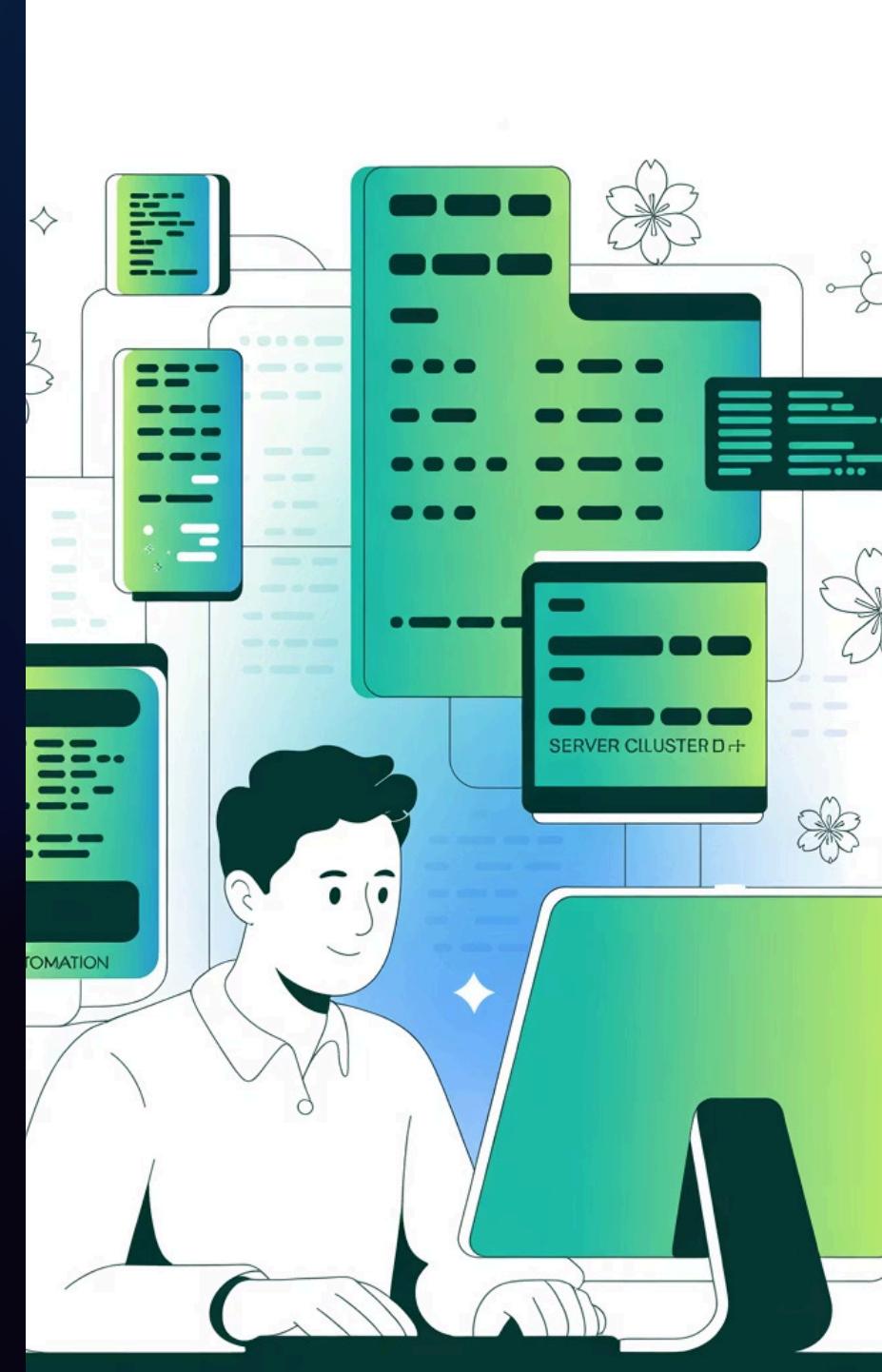
Atualizar um member por vez para garantir disponibilidade contínua durante atualizações de aplicações ou do próprio Liberty.

## Monitoramento

Integrar com Prometheus, Grafana para visualização em tempo real de métricas de performance e disponibilidade.

## Backup de Configuração

Sincronizar server.xml e artefatos em repositório Git para rastreabilidade e recuperação rápida.



# Desafios Comuns



## Desbalanceamento de tráfego

→ Afinidade no LB para sessões stateful



## Conflito de portas

→ Usar host="\*" no httpEndpoint



## Overhead de replicação

→ Minimizar objetos grandes em sessão



## Gerenciamento manual

→ Usar scripts para reduzir erro humano

A maioria dos problemas em clusters Liberty tem origem em configurações incorretas de rede e sessão. Implementar monitoramento proativo é essencial para identificar estes problemas antes que afetem os usuários.

# Configuração de Cluster Tradicional (Sessões e HA)



## Ativar Cache de Sessões

Adicione a feature `sessionCache-1.0` no `server.xml` e configure provedores como Infinispan para replicação distribuída.

## Integrar Load Balancer

Configure Apache ou NGINX para distribuição de tráfego entre os nós do cluster.

## Testar Failover

Implante uma aplicação de teste, acesse via balancer e simule falha em um nó para validar o failover automático.

## Persistência para Apps Stateful

Use bancos de dados externos para armazenar estados críticos da aplicação.

Esta configuração assegura alta disponibilidade e continuidade de sessões mesmo durante falhas de nós individuais.



# Configuração em Kubernetes com Open Liberty Operator

## Instalação do Operator

- Via OperatorHub no OpenShift
- Ou com `kubectl apply -f operator.yaml`

## Principais Benefícios

- Automação do gerenciamento de deployments
- Facilidade de escalabilidade
- Configuração declarativa via YAML

## Custom Resource YAML

```
apiVersion: apps.openliberty.io/v1
kind: OpenLibertyApplication
metadata:
  name: minha-aplicacao
spec:
  applicationImage:
    myapp:latest
  replicas: 3
  service:
    type: ClusterIP
```

Esta configuração facilita setups cloud-native com poucos comandos.

# Configuração em Kubernetes (Auto-Scaling e HA)



## Auto-scaling com HPA

Ative no YAML com:

```
autoscaling:  
  minReplicas: 2  
  maxReplicas: 10
```

```
targetCPUUtilizationPercentage:  
  80
```



## Persistência de Sessões

Configure persistência com:

```
volumeClaims:  
  - name: sessions-data  
    mountPath: /output/sessions
```



## Monitoramento

Integre com Prometheus:

```
monitoring:  
  labels:  
    app: minha-aplicacao
```

Estas funcionalidades promovem alta disponibilidade dinâmica em ambientes Kubernetes, adaptando-se automaticamente às mudanças de carga e condições de infraestrutura.

# Best Practices para Configuração (Parte 1)



01

## **Image Streams para Updates**

Use Image Streams para atualizações rolling sem interrupções, garantindo zero downtime durante deployments.

02

## **MicroProfile Health**

Ative com `livenessProbe` e `readinessProbe` no YAML para verificações automáticas de saúde da aplicação.

03

## **Segurança Delegada**

Deleve SSO via Operator e use cert-manager para gerenciamento automatizado de certificados TLS.

04

## **Automação**

Priorize automação de processos de build, teste e deploy para maior eficiência operacional.

# Best Practices para Configuração (Parte 2)

## Limites de Recursos

Defina limites claros por pod:

```
resources:  
  requests:  
    cpu: 1  
    memory: 2Gi  
  limits:  
    cpu: 2  
    memory: 4Gi
```

Isto evita overcommit e garante performance previsível.

## Persistent Volumes

Use volumes persistentes para:

- Backups de configuração
- Armazenamento de logs
- Cache de aplicação

Teste regularmente planos de disaster recovery.

## Integração de Serviços

Configure integrações automáticas:

- Bancos de dados via Operator
- Serviços de cache distribuído
- Sistemas de mensageria

Estas práticas otimizam performance e manutenção.

# Desafios Comuns e Soluções

## Desbalanceamento de carga

Solucionar com [regras de afinidade](#) em Kubernetes para distribuição apropriada de tráfego.

## Conflitos de porta

Configure `host="*"` em `httpEndpoint` para evitar conflitos e permitir bind em múltiplas interfaces.

## Overhead de cluster

Otimizar com features mínimas, ativando apenas o necessário para sua aplicação específica.

## Troubleshooting via logs

Use ferramentas como Kibana para análise centralizada de logs e diagnóstico rápido.



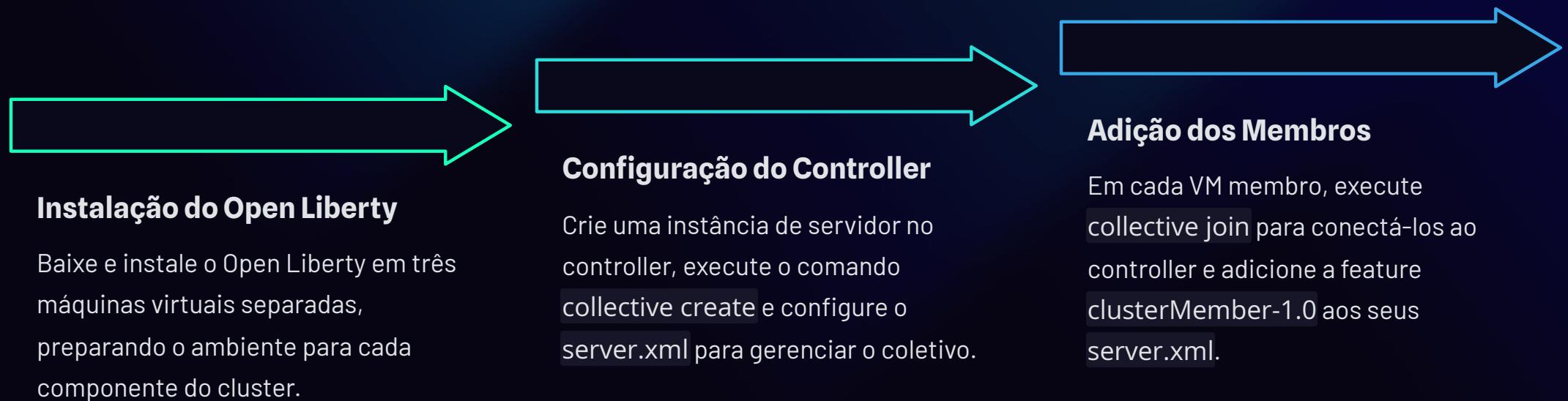
## Conclusão

Estas soluções mitigam problemas frequentes em ambientes de cluster, mantendo sua infraestrutura Liberty estável e resiliente.

Lembre-se: [monitore proativamente](#) para identificar problemas antes que afetem seus usuários.

# Laboratório 1: Configuração Básica de um Cluster Tradicional (Coletivo)

Neste laboratório, você construirá um cluster Open Liberty simples, composto por um controller e dois membros, utilizando VMs.



- ⓘ **Dicas:** Utilize as portas padrão do Liberty para simplificar a configuração. Após cada passo, verifique o status do servidor com `./server status` para confirmar a operação.