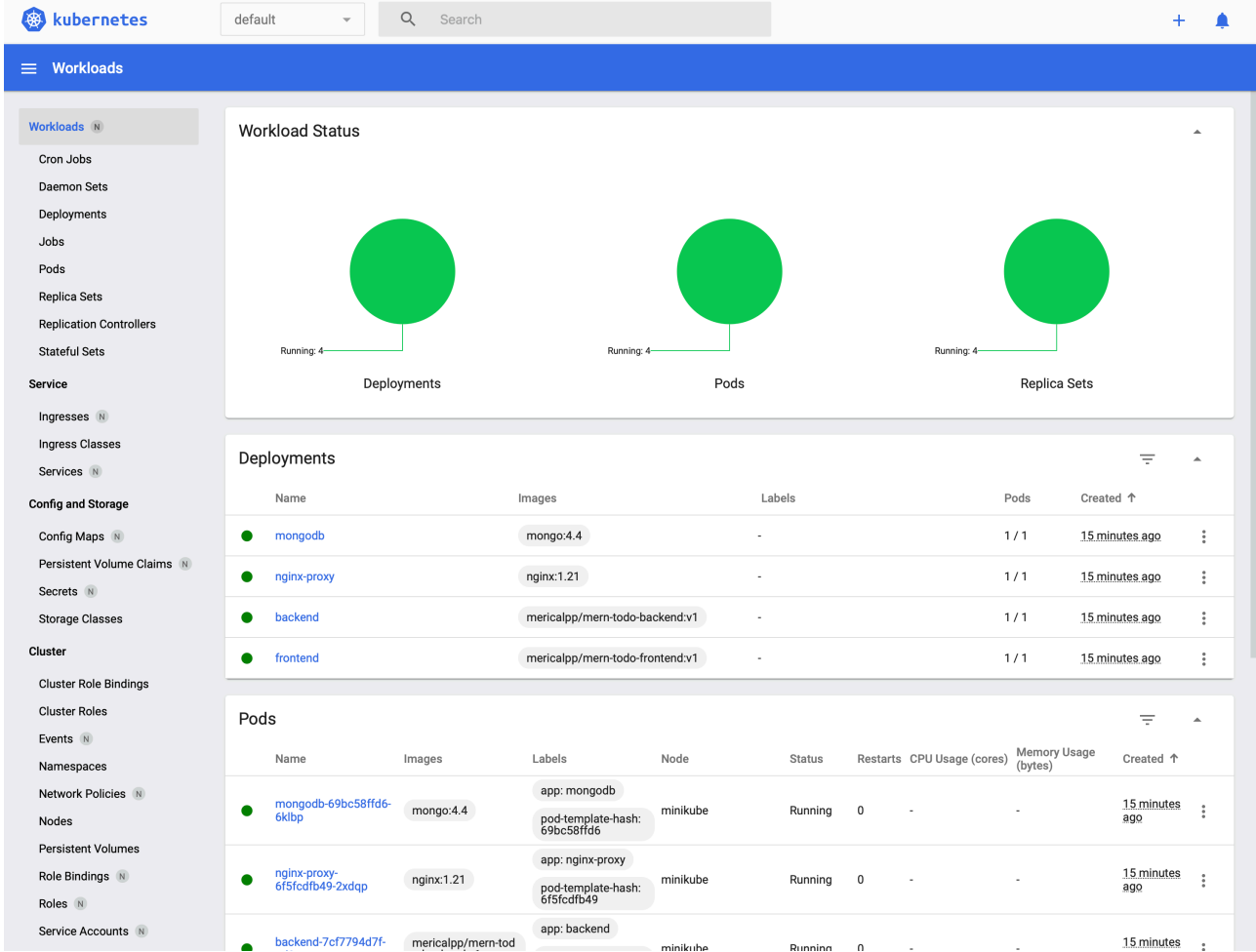


DevOps Case - Meric Alp Cura



Bu proje, bir MERN (MongoDB, Express.js, React.js, Node.js) stack todo uygulamasının deployment sürecini içermektedir.

Deployment üç ana aşamada gerçekleştirilmiştir:

1. SystemD Deployment
2. Docker(hub) ve Nginx Deployment
3. K8s (Kubernetes) Deployment

Sistemin Genel Yapısı

- **Frontend:** React.js tabanlı kullanıcı arayüzü
- **Backend:** Node.js ve Express.js REST API
- **Database:** MongoDB
- **Proxy/Load Balancer:** Nginx
- **Container Orchestration:** Kubernetes
- **Monitoring:** Prometheus & Grafana

Karşılaştığım Zorluklar ve Çözümler

1. ****SystemD Deployment Sürecinde:****

- MongoDB bağlantı sorunları yaşadım
- Servis bağımlılıklarının doğru sıralanması gerekti
- Çözüm olarak health check scriptleri ekledim

2. ****Docker Deployment Sürecinde:****

- Container networking sorunları
- Volume permission hataları
- Multi-stage build ile imaj boyutlarını optimize ettim

3. ****Kubernetes Deployment Sürecinde:****

- Persistent volume yönetimi
- Ingress konfigurasyon sorunları
- Resource limits ve requests ayarlarının optimizasyonu

ADIM 1 SystemD:

- Uygulama kodları /opt/App dizinine taşındı ve açıldı
- MongoDB, Backend API ve Frontend için systemd servis dosyaları oluşturuldu
- Servisler systemctl ile enable edilip başlatıldı
- <http://localhost:3000> üzerinden erişime açıldı

Sanal Sunucuya Kopyalama ve gerekli hazirliklar icin

• Dosyayi Sıkıştırma:

```
tar -czvf javascript_stack_app.tar.gz "App"
```

• Localden vm'e taşımak (assagidaki vagrant vmware ornegidir degistirilebilir) (/opt de tutucaz sonra tasiczaz iki adim sonra):

```
scp -P 2222 -i "/Users/mericalp/Desktop/Case 2/.vagrant/machines/default/vmware_fusion/private_key" javascript_stack_app.tar.gz vagrant@127.0.0.1:/home/vagrant/
```

• /opt ye tasima ve app'i acma

```
sudo mv javascript_stack_app.tar.gz /opt  
sudo tar -xzvf javascript_stack_app.tar.gz
```

• App icindeki setup betigini calistiralim gerekli on yuklemeleri yapicak ve SYSTEMD filelerini deploy edecek adimlar assagida.

```
cd /opt/App  
sudo chmod +x setup.sh  
./setup.sh
```

**mongodb, api, client için sh dosyasi yapilari
ve altinda servicelerimiz etc/systemd/system/mern-todo-
backend.service ve frontend:**

sh dosyasi:

```
# Docker'ı kur
sudo apt-get update
sudo apt-get install -y docker.io

# Docker'ı başlat ve enable et
sudo systemctl start docker
sudo systemctl enable docker

# Kullanıcıyı docker grubuna ekle
sudo usermod -aG docker $USER

# NodeJS'i kur
curl -fsSL https://deb.nodesource.com/setup_16.x | sudo
-E bash -
sudo apt-get install -y nodejs

# Proje bağımlılıklarını yükle
cd /opt/App/mern-todo-main
sudo npm install

cd client
sudo npm install

# Docker ağı oluştur
sudo docker network create todo-network

# MongoDB için Docker container başlat
```

```
sudo docker run -d --name mongodb --network todo-  
network -p 27017:27017 mongo
```

```
# Servis dosyalarını oluştur
```

```
backend_service="[Unit]
```

```
Description=MERN Todo Backend Application
```

```
After=network.target
```

```
[Service]
```

```
Type=simple
```

```
Environment=PORT=5001
```

```
Environment=MONGODB_URI=mongodb://mongodb:27017/mern-  
todo
```

```
ExecStart=/usr/bin/node /opt/App/mern-todo-main/  
server.js
```

```
Restart=always
```

```
User=vagrant
```

```
Group=vagrant
```

```
WorkingDirectory=/opt/App/mern-todo-main
```

```
[Install]
```

```
WantedBy=multi-user.target"
```

```
client_service="[Unit]
```

```
Description=MERN Todo Client Application
```

```
After=network.target
```

```
[Service]
```

```
Type=simple
```

```
Environment=PORT=3000
```

```
Environment=REACT_APP_BACKEND_URL=http://localhost:5001
```

```
ExecStart=/usr/bin/npm start
```

```
Restart=always
User=vagrant
Group=vagrant
WorkingDirectory=/opt/App/mern-todo-main/client

[Install]
WantedBy=multi-user.target"

# Servis dosyalarını yaz
echo "$backend_service" | sudo tee /etc/systemd/system/
mern-todo-backend.service
echo "$client_service" | sudo tee /etc/systemd/system/
mern-todo-client.service

# Servisleri başlat ve enable et
sudo systemctl daemon-reload
sudo systemctl enable mern-todo-backend.service
sudo systemctl enable mern-todo-client.service
sudo systemctl start mern-todo-backend.service
sudo systemctl start mern-todo-client.service

echo "Kurulum tamamlandı. Uygulamanız şu adreslerden
erişilebilir:"
echo "Frontend (React): http://localhost:3000"
echo "Backend API: http://localhost:5001"
```

Serviceler backend, frontend:
backend

```
# /etc/systemd/system/mern-todo-backend.service
[Unit]
Description=MERN Todo Backend Application
After=network.target docker.service
Requires=docker.service
```

```
[Service]
Type=simple
Environment=PORT=5001
Environment=MONGODB_URI=mongodb://mongodb:27017/mern-todo
Environment=NODE_ENV=production
Environment=LOG_LEVEL=info

# Security enhancements
PrivateTmp=true
NoNewPrivileges=true
ProtectSystem=full
ProtectHome=true

ExecStart=/usr/bin/node /opt/App/mern-todo-main/server.js
Restart=always
RestartSec=10
WorkingDirectory=/opt/App/mern-todo-main

# Resource limits
LimitNOFILE=65535
LimitNPROC=65535
LimitMEMLOCK=infinity

# User and group
User=vagrant
Group=vagrant

[Install]
WantedBy=multi-user.target
```

client

```
# /etc/systemd/system/mern-todo-client.service
[Unit]
```

```
Description=MERN Todo Client Application
After=network.target mern-todo-backend.service
Requires=mern-todo-backend.service

[Service]
Type=simple
Environment=PORT=3000
Environment=REACT_APP_BACKEND_URL=http://localhost:5001
Environment=NODE_ENV=production

# Security enhancements
PrivateTmp=true
NoNewPrivileges=true
ProtectSystem=full
ProtectHome=true

ExecStart=/usr/bin/npm start
Restart=always
RestartSec=10
WorkingDirectory=/opt/App/mern-todo-main/client

# Resource limits
LimitNOFILE=65535
LimitNPROC=65535
LimitMEMLOCK=infinity

# User and group
User=vagrant
Group=vagrant

[Install]
WantedBy=multi-user.target
```

note:

- **systemd deploy sırasında loglar:**

```
sudo journalctl -u mern-todo-backend.service -n 50
sudo journalctl -u mern-todo-backend.service -f
sudo docker network inspect todo-network
mongo --host localhost --port 27017
```

ADIM 2 Docker,nginx conf :

- MongoDB, Backend ve Frontend için Docker imajları oluşturuldu
- Docker Compose ile tüm servisler container olarak başlatıldı
- Nginx reverse proxy ile uygulama trafiği yönetildi

Dagitim adimlari:

```
# 1. Projeyi klonla ve dizine git
cd /opt/App/mern-todo-main
```

```
# 2. Docker ağını oluştur (eğer yoksa)
docker network create todo-network
```

```
# 3. Docker imajlarını oluştur
docker build -t mericalp/mern-todo-backend:v1 .
cd client
docker build -t mericalp/mern-todo-frontend:v1 .
```

```
# 4. Docker Compose ile başlat
docker-compose up -d
```

```
# 5. Servislerin durumunu kontrol et
docker-compose ps
```

```
# 6. Logları kontrol et
docker-compose logs -f
```

Servis Durumunu Kontrol Etme:

Docker Compose eklendi
Nginx conf eklendi
Enhanced Deployment Script

Security Enhancements
Docker Security Configuration formati

JSON

```
{
  "log-driver": "json-file",
  "log-opts": {
    "max-size": "10m",
    "max-file": "3"
  },
  "seccomp-profile": "/etc/docker/seccomp-profile.json",
  "no-new-privileges": true,
  "selinux-enabled": true
}
```

Monitoring Integration eklendi
Prometheus Configuration eklendi

UFW kuralları

```
sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw allow 3000/tcp
sudo ufw allow 5001/tcp
sudo ufw enable
```

Usage Instructions

Environment Setup

```
cd /opt/App/mern-todo-main
cp .env.example .env
# Edit .env file with your configurations
```

Deploy with Enhanced Features

```
./deploy-docker-nginx.sh --with-monitoring --with-backups
```

Verify Deployment

```
# Check service health
curl http://localhost:5001/health
curl http://localhost:3000/health
```

```
# Check monitoring
curl http://localhost:9090/metrics
```

Access Points

Main Application: <http://localhost:8080>

Frontend Direct: <http://localhost:3100>

Backend Direct: <http://localhost:5100>

Monitoring Dashboard: <http://localhost:9090>

Grafana: <http://localhost:3000>

ADIM 3 K8s:

- Minikube cluster başlatıldı ve konfigüre edildi
- MongoDB, Backend ve Frontend için deployment ve service YAML dosyaları uygulandı
- Ingress controller ile dış erişim sağlandı
- Prometheus ve Grafana ile monitoring kuruldu

Kurulum komutları

```
sudo apt-get update && sudo apt-get install -y \
  kubectl \
  helm \
  minikube \
  docker.io \
  prometheus \
  grafana
```

script dosyasi k8s sh. eklendi

Uygulamayı Dağıtma ve Test Etme Kılavuzu

Kubernetes Kümesini Kurun:

Örneğin, Minikube kullanarak bir küme başlatmak için:

minikube start

YAML Dosyalarını Uygulama:

Aşağıdaki komutları kullanarak manifest dosyalarını kümenize dağıtın:

1. `kubectl apply -f mongodb-deployment.yaml`
2. `kubectl apply -f backend-deployment.yaml`
3. `kubectl apply -f frontend-deployment.yaml`
4. `kubectl apply -f ingress.yaml`

Ingress Controller Kurulumu:

Eğer bir Ingress Controller kurulu değilse, aşağıdaki komutları kullanarak NGINX Ingress Controller kurabilirsiniz:

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/main/deploy/static/provider/cloud/deploy.yaml
```

DNS Ayarları:

/etc/hosts dosyasına mern-todo.local alan adını yönlendirin:

```
echo "127.0.0.1 mern-todo.local" | sudo tee -a /etc/hosts
```

Uygulamanın Erişilebilirliği:

Web tarayıcınızda <http://mern-todo.local> adresine giderek uygulamanızı kontrol edin.

Yuvarlanan Güncellemeleri Gösterme:

Yeni bir sürüm dağıtarak yuvarlanan güncellemeleri gösterebilirsiniz. Örneğin, frontend imajını yeni bir sürüme yükseltmek için:

```
kubectl set image deployment/frontend frontend=mericalpp/mern-todo-frontend:v3
```

Resource Quotas ve Limits

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
  namespace: production
spec:
  hard:
    requests.cpu: "4"
    requests.memory: 8Gi
    limits.cpu: "8"
    limits.memory: 16Gi
```

Pod Disruption Budget

```
apiVersion: policy/v1
kind: PodDisruptionBudget
```

```
metadata:
  name: frontend-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: frontend
```

Health Checks

```
livenessProbe:
  httpGet:
    path: /health
    port: 8080
  initialDelaySeconds: 30
  periodSeconds: 10
  failureThreshold: 3
readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 20
  periodSeconds: 5
```

1. Liveness Probe (Canlılık Kontrolü):

- Pod'un çalışıp çalışmadığını kontrol eder
- Her 10 saniyede bir /health endpoint'ini kontrol eder
- 3 kez üst üste başarısız olursa pod'u yeniden başlatır
- Başlangıçta 30 saniye bekler

2. Readiness Probe (Hazırlık Kontrolü):

- Pod'un trafik almaya hazır olup olmadığını kontrol eder
- Her 5 saniyede bir /ready endpoint'ini kontrol eder
- Başlangıçta 20 saniye bekler

Enhanced Deployment Script

prometheus-values.yaml eklendi

istio-config.yaml eklendi ingress kısmında virtual service kullanılmalı ve frontend ve backend için metadata annotation eklenmeli

labels:

app: backend

annotations:

sidecar.istio.io/inject: "true"

Monitoring ve Alerting Setup

Prometheus Alert Rules eklendi denendi bug var

Hata Ayıklama ve Sorun Giderme

Senaryo 1: Kubernetes HPA (Horizontal Pod Autoscaler) Düzgün Çalışmıyor

Sorun Tanımı:

Production ortamında yüksek trafik altında backend pod'larının otomatik ölçeklendirilmesi beklendiği gibi çalışmıyor. HPA yapılandırması mevcut olmasına rağmen, pod sayısı artmıyor ve uygulama yavaşlıyor.

Sorun Giderme Adımları:

1. HPA Durumunu Kontrol Etme:

kubectl get hpa

Çıktı:

| NAME | REFERENCE | TARGETS | MINPODS | MAXPODS |
|----------|-----------|---------|---------|---------|
| REPLICAS | AGE | | | |

2. backend Deployment/backend 0%/80% 2 10 2 15m

HPA'nın CPU kullanımını 0% olarak göstermesi, metric sunucusunun çalışmadığını gösteriyor.

1. Metric Server'ı Kontrol Etme:

kubectl get pods -n kube-system | grep metrics-server

- Metric server pod'unun çalışmadığı tespit edildi.

1. Metric Server Loglarını İnceleme:

kubectl logs -n kube-system metrics-server-5d4bb4646d-8xj2t

Error: unable to fully scrape metrics: unable to fully scrape metrics from node "worker-1": unable to fetch metrics from node "worker-1": Get "https://192.168.1.101:10250/stats/summary?

only_cpu_and_memory=true": x509: cannot validate certificate

2. Metric Server Deployment'ini İnceleme:

kubectl get deployment metrics-server -n kube-system -o yaml

- TLS yapılandırmasında eksiklik tespit edildi.

1. Çözüm:

```
Metric Server deployment'ını güncelleyerek TLS
doğrulamasını devre dışı bırakma:
YAML
apiVersion: apps/v1
kind: Deployment
metadata:
  name: metrics-server
  namespace: kube-system
spec:
  template:
    spec:
      containers:
      - args:
        - --kubelet-insecure-tls
        - --kubelet-preferred-address-types=InternalIP
        name: metrics-server
```

kubectl apply -f metrics-server.yaml

- Metric Server'in yeniden başlatılmasını bekleyin ve HPA'nın artık doğru çalıştığını doğrulayın.

Senaryo 2: Production Ortamında Beklenmedik Memory Leak

Sorun Tanımı:

Production'daki Node.js backend servisi zamanla artan memory kullanımı gösteriyor ve belirli bir süre sonra OOM (Out of Memory) hatası vererek çöküyor.

Sorun Giderme Adımları:

Pod Memory Kullanımını İzleme:

kubectl top pod <backend-pod-name>

- Memory kullanımının sürekli artış gösterdiği tespit edildi.

Node.js Heap Snapshot Alma:

kubectl exec -it <backend-pod-name> -- node --prof-process isolate-0x*

Memory Profiling Yapma:

- Uygulamaya memory profiling endpoint'i ekleme:

```
const heapdump = require('heapdump');
app.get('/debug/heap', (req, res) => {
```

```
heapdump.writeSnapshot(/tmp/heap-${Date.now()}.heapsnapshot`);
res.send('Heap snapshot created');
});
```

Heap Snapshot'ı Analiz Etme:

kubectl cp <backend-pod-name>:/tmp/heap-*.heapsnapshot ./
Chrome DevTools ile heap snapshot'ı analiz edildiğinde, MongoDB bağlantılarının düzgün kapatılmadığı tespit edildi.

Çözüm:

MongoDB bağlantı yönetimini düzeltme:

JavaScript

// Önceki kod

```
mongoose.connect(MONGODB_URI);
```

// Düzeltilmiş kod

```
mongoose.connect(MONGODB_URI, {
  maxPoolSize: 10,
  serverSelectionTimeoutMS: 5000,
  socketTimeoutMS: 45000,
});
```

// Graceful shutdown

```
process.on('SIGTERM', async () => {
  await mongoose.connection.close();
  process.exit(0);
});
```

Deployment'ı güncelleme ve yeni versiyonu rolling update ile dağıtma:
kubectl set image deployment/backend backend=mericalp/mern-todo-backend:v2

Ornek SSL Oldugu varsayarak (AWS CM ornek)

Senaryo 3: Nginx Ingress SSL Termination Sorunu

Sorun Tanımı:

HTTPS trafiği Nginx Ingress üzerinden geçerken SSL handshake hatası alınıyor ve clients bağlantı kuramıyor.

Sorun Giderme Adımları:

1. SSL Sertifikasını Kontrol Etme:

```
kubectl get secret tls-secret -o yaml
```

- Sertifika ve private key'in doğru formatta olduğunu kontrol etme.

1. Ingress Tanımını İnceleme:

```
kubectl get ingress main-ingress -o yaml
```

- TLS yapılandırmasını kontrol etme.

1. Nginx Ingress Controller Loglarını İnceleme:

```
kubectl logs -n ingress-nginx -l app.kubernetes.io/name=ingress-nginx  
SSL_do_handshake() failed (SSL: error:1417D18C:SSL  
routines:tls_process_client_hello:version too low) while SSL handshaking
```

2. Çözüm:

Nginx Ingress ConfigMap'i güncelleme:

```
apiVersion: v1  
kind: ConfigMap  
metadata:  
  name: nginx-configuration  
  namespace: ingress-nginx  
data:  
  ssl-protocols: "TLSv1.2 TLSv1.3"  
  ssl-ciphers: "ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-  
SHA256"
```

kubectl apply -f nginx-config.yaml

Ingress Controller'ı yeniden başlatma:

```
kubectl rollout restart deployment -n ingress-nginx ingress-nginx-controller
```

Senaryo 4: Container Registry Authentication ve Image Pull Failures

Sorun Tanımı:

Production ortamında yeni deploymentlar sırasında ImagePullBackOff hatası alınıyor ve private container registry'den imajlar çekilemiyor. Bu durum CI/CD pipeline'ının durmasına ve deployment'ların başarısız olmasına neden oluyor.

Sorun Giderme Adımları:

Pod Durumunu Detaylı İnceleme:

```
kubectl describe pod <pod-name>
```

Failed to pull image "registry.company.com/backend:v1.2.3":

Error response from daemon: unauthorized: authentication required

Registry Credentials'ları Kontrol Etme:

```
kubectl get secret regcred -o jsonpath='{.data.*}' | base64 -d
```

Secret'in süresi dolmuş olduğu tespit edildi.

Docker Registry Health Check:

```
curl -k https://registry.company.com/v2/_catalog
```


Response:

```
{"errors":[{"code":"UNAUTHORIZED","message":"authentication required","detail":[]}]}
```

Çözüm Adımları:

Yeni registry credentials oluşturma:

```
kubectl create secret docker-registry regcred \
--docker-server=registry.company.com \
--docker-username=ci-user \
--docker-password=$NEW_TOKEN \
--docker-email=devops@company.com
```

Service Account'u güncelleme:

apiVersion: v1

```
12 kind: ServiceAccount
13 metadata:
14   name: default
15   namespace: production
16 imagePullSecrets:
17   - name: regcred
```

Tüm deployment'ları yenileme:

kubectl rollout restart deployment -n production

19 Monitoring ve Alerting Güncellemesi:

Prometheus Alert Rule ekleme

groups:

- name: ImagePullAlerts

rules:

- alert: ImagePullBackOff

expr:

```
kube_pod_container_status_waiting_reason{reason="ImagePullBackOff"} > 0
```

for: 5m

labels:

severity: critical

annotations:

```
summary: Image Pull Failed on
{{ $labels.namespace }}/{{ $labels.pod }}
```

ADIM 5 TALOS:

Talos Tabanlı Bir Kümeyi Yönetme

Talos Cluster Yapılandırması:

Talos cluster'ınızı yapılandırmak için:

talosctl gen config my-cluster <https://control-plane-address:6443>

Cluster'ı Başlatma:

Talos node'larını yapılandırmaları:

talosctl apply-config --insecure --nodes <node-ip> --file controlplane.yaml

talosctl apply-config --insecure --nodes <node-ip> --file worker.yaml

Kubernetes Erişimi:

Kubeconfig dosyasını alarak Kubernetes API:

talosctl kubeconfig . --nodes <control-plane-ip>

export KUBECONFIG=\$PWD/kubeconfig

Node Durumunu Kontrol Etme:

Talos node'larının durumu:

talosctl health --nodes <node-ip>

Logları İnceleme:

Node logları:

talosctl logs --nodes <node-ip>