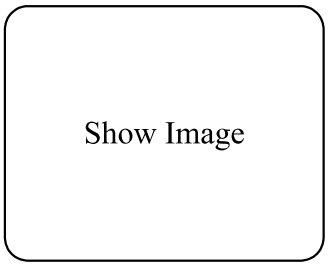
Day 3 - Cloud-Native Telecom Lab Guide

Perfect Training Center - Orchestration & Automation Hands-On



📋 Lab Overview

- **Duration:** 30-35 minutes
- **Difficulty:** Intermediate
- **Focus:** Service orchestration, automation, DevOps
- **Requirements:** Modern browser, internet connection

🎯 Lab Objectives

By completing this lab, you will:

1. Design a telecom service using orchestrator
2. Deploy VNFs automatically
3. Configure auto-scaling policies
4. Test self-healing capabilities
5. Monitor service health
6. Experience zero-touch operations

🔬 Main Lab Exercise: Service Orchestration

Part 1: Access the Orchestrator (5 minutes)

Step 1: Login to Platform

1. **Open browser** (Chrome/Firefox recommended)
2. **Navigate to:** `orchestrator.perfect-tc.com`

3. **Select:** English or العربية

Step 2: Authentication

```
Username: student@perfect-tc
Password: CloudNative2024!
Workspace: Telecom-Lab-3
```

Step 3: Explore Interface

- **Service Designer -** Drag & drop VNFs
- **Catalog -** Pre-built templates
- **Monitoring -** Real-time dashboards
- **Automation -** Policy engine

📸 **Screenshot Point:** Capture the orchestrator dashboard

Part 2: Design Your Service (10 minutes)

Step 1: Create New Service

1. Click **"New Service"**
2. Name: `IMS-CloudNative-[YourName]`
3. Description: `Virtual IP Multimedia Subsystem`
4. Version: `1.0`

Step 2: Add VNFs (Virtual Network Functions)

Drag these VNFs from catalog to canvas:

1. P-CSCF (Proxy-Call Session Control Function)

```yaml
VNF: P-CSCF
Version: 2.0
Instances: 2 (for redundancy)
Resources:
  vCPU: 4
  RAM: 8GB
  Storage: 20GB
```

2. I-CSCF (Interrogating-CSCF)

```yaml
VNF: I-CSCF
Version: 2.0
Instances: 2
Resources:
  vCPU: 4
  RAM: 8GB
  Storage: 20GB
```

3. S-CSCF (Serving-CSCF)

```yaml
VNF: S-CSCF
Version: 2.0
Instances: 3 (main service node)
Resources:
  vCPU: 8
  RAM: 16GB
  Storage: 50GB
```

4. HSS (Home Subscriber Server)

```yaml
VNF: HSS
Version: 3.0
Instances: 2
Resources:
  vCPU: 8
  RAM: 32GB
  Storage: 100GB
```

Step 3: Connect VNFs

Draw connections between VNFs:

1. **Users → P-CSCF** (SIP interface)
2. **P-CSCF → I-CSCF** (Internal)
3. **I-CSCF → S-CSCF** (Internal)
4. **S-CSCF → HSS** (Diameter)

📸 **Screenshot Point:** Capture your service design

Step 4: Configure Networks

Add networks:

1. **Management Network -** All VNFs
2. **Signaling Network -** All VNFs
3. **Media Network -** P-CSCF only

Part 3: Configure Automation Policies (8 minutes)

Step 1: Auto-Scaling Policy

Click **"Policies" → "Add Policy"**

Scale-Out Policy:

```yaml
Name: ScaleOut-PCSCF
Type: Threshold
Metric: CPU_Utilization
Threshold: >75%
Duration: 5 minutes
Action: Add 1 instance
Cooldown: 10 minutes
Max_Instances: 5
```

Scale-In Policy:

```yaml
Name: ScaleIn-PCSCF
Type: Threshold
Metric: CPU_Utilization
Threshold: <25%
Duration: 10 minutes
Action: Remove 1 instance
Cooldown: 15 minutes
Min_Instances: 2
```

Step 2: Self-Healing Policy

```yaml
```

```
Name: AutoHeal-AllVNFs
Type: Health_Check
Check_Interval: 30 seconds
Failure_Threshold: 3
Action: Restart_VNF
Escalation: Recreate_if_restart_fails
```

Step 3: Load Balancing

```yaml
Name: LoadBalance-PCSCF
Type: Round_Robin
Health_Check: HTTP
Check_Path: /health
Interval: 10 seconds
```

📸 **Screenshot Point:** Document your policies

Part 4: Deploy the Service (7 minutes)

Step 1: Validate Design

1. Click **"Validate"**
2. Fix any errors (red highlights)
3. Review warnings (yellow)

Step 2: Deploy Service

1. Click **"Deploy"** button
2. Select environment: `Production-Cloud`
3. Deployment options:
   - ✓ Enable monitoring
   - ✓ Enable logging
   - ✓ Auto-start services
4. Click **"Start Deployment"**

Step 3: Watch the Magic! 🎯

Monitor real-time deployment:

1. **Resource Allocation** - VMs spinning up
2. **Network Configuration** - SDN magic
3. **VNF Installation** - Containers deploying
4. **Service Activation** - Going live
5. **Health Checks** - All green!

Timeline:

```
0:00 - Deployment started
0:30 - Resources allocated
1:00 - Networks configured
2:00 - VNFs deployed
3:00 - Services activated
3:30 - Health checks passed
4:00 - Service operational! ✅
```

📸 **Screenshot Point:** Capture deployment progress

Part 5: Test Automation Features (10 minutes)

Test 1: Auto-Scaling

1. Go to **"Testing"** tab
2. Click **"Traffic Generator"**
3. Configure:

```
Target: P-CSCF
Load: Start 100 → Ramp to 1000 calls/sec
Duration: 5 minutes
```

4. Click **"Start Test"**
5. Watch:
   - CPU usage climbing
   - Auto-scaling triggers at 75%
   - New instance spawning
   - Load rebalancing

Expected: P-CSCF scales from 2 to 3 instances

Test 2: Self-Healing

1. Go to **"Chaos Testing"**
2. Select: S-CSCF Instance 1
3. Action: **"Kill Process"**
4. Watch:
   - Health check fails
   - Self-healing triggers
   - VNF restarts automatically
   - Service continues uninterrupted

Expected: Service heals in <60 seconds

Test 3: Load Balancing

1. Go to **"Monitoring"**
2. View P-CSCF instances
3. Check request distribution:
   - Instance 1: ~33%
   - Instance 2: ~33%
   - Instance 3: ~34%

📸 **Screenshot Point:** Capture all test results

Part 6: Monitor & Optimize (5 minutes)

Step 1: Dashboard Overview

Navigate to monitoring dashboard:

- **Service Health:** All green
- **Performance Metrics:**
  - Latency: <20ms
  - Throughput: 1000 calls/sec
  - Success Rate: 99.9%
- **Resource Usage:**
  - CPU: 45% average

- Memory: 60% average
- Network: 100Mbps

Step 2: Logs & Analytics

1. Click **"Logs"**
2. Filter: `Last 5 minutes`
3. See:
   - Deployment logs
   - Scaling events
   - Healing actions
   - Performance metrics

Step 3: Cost Analysis

View **"Cost Dashboard"**:

```
Resources Used:
- vCPU: 32 cores
- RAM: 128 GB
- Storage: 380 GB
- Network: 1 Gbps

Hourly Cost: $12.50
Monthly (projected): $9,000
Traditional Cost: $45,000
Savings: 80%! 💰
```

🧬 Bonus Lab: GitOps Workflow

Create Infrastructure as Code (10 minutes)

Step 1: Access Git Repository

1. Go to: `git.perfect-tc.com`
2. Login with lab credentials
3. Fork: `telecom-iac-template`

Step 2: Define Service in YAML

Create file: services/ims-service.yaml

```yaml
```

```yaml
apiVersion: nfv.io/v1
kind: NetworkService
metadata:
  name: ims-cloudnative
  namespace: production
spec:
  version: "1.0"
  vendor: "Perfect-TC"
  vnfs:
    - name: pcscf
      type: P-CSCF
      version: "2.0"
      replicas: 2
      resources:
        cpu: "4"
        memory: "8Gi"
        storage: "20Gi"
      scaling:
        min: 2
        max: 5
        cpu_threshold: 75
    - name: icscf
      type: I-CSCF
      version: "2.0"
      replicas: 2
      resources:
        cpu: "4"
        memory: "8Gi"
        storage: "20Gi"
    - name: scscf
      type: S-CSCF
      version: "2.0"
      replicas: 3
      resources:
        cpu: "8"
        memory: "16Gi"
        storage: "50Gi"
    - name: hss
      type: HSS
      version: "3.0"
      replicas: 2
      resources:
        cpu: "8"
```

```
      memory: "32Gi"
      storage: "100Gi"
  policies:
    autoScaling: enabled
    selfHealing: enabled
    monitoring: prometheus
```

Step 3: Commit and Deploy

```bash
# Add files
git add services/ims-service.yaml

# Commit with message
git commit -m "Deploy IMS service - cloud native"

# Push to trigger deployment
git push origin main
```

Step 4: Watch GitOps Magic

1. Go to **"GitOps Dashboard"**
2. See:
   - Git commit detected
   - Validation running
   - Approval workflow
   - Automatic deployment
   - Service live!

📸 **Screenshot Point:** Capture GitOps flow

🛠️ Additional Exercises

Exercise 1: Create Network Slice

Design and deploy a network slice for:

- **Gaming:** Ultra-low latency (<5ms)
- **IoT:** Massive connections (10k devices)
- **Video:** High bandwidth (100Mbps)

Exercise 2: Intent-Based Configuration

Use intent engine:

1. Type: "I need five-nines uptime for VoIP"
2. Watch translation to:
   - Redundancy policies
   - Health checks
   - Failover configuration

Exercise 3: AI-Powered Optimization

1. Enable **"AI Optimizer"**
2. Run for 5 minutes
3. See recommendations:
   - Resource right-sizing
   - Cost optimization
   - Performance improvements

🐛 Troubleshooting Guide

Common Issues:

"Cannot access orchestrator"

- Check internet connection
- Try incognito mode
- Clear browser cache
- Use backup URL: orchestrator2.perfect-tc.com

"VNF deployment fails"

- Check resource availability
- Verify network configuration
- Review error logs
- Reduce resource requests

"Auto-scaling not working"

- Verify policy syntax
- Check metrics collection
- Ensure cooldown expired
- Test with lower threshold

"Service unhealthy"

- Check individual VNF status
- Verify network connectivity
- Review configuration
- Restart deployment

📊 Lab Report Template

Complete and submit:

```markdown
```

Lab 3 Report: Cloud-Native Orchestration

Name: _____
Date: _____

Service Design
- Service Name: _____
- VNFs Deployed: _____
- Total Resources: _____

Automation Tests
Auto-Scaling
- Trigger Threshold: _____%
- Scale-Out Time: _____ seconds
- New Instances: _____

Self-Healing
- Failure Injection: _____
- Recovery Time: _____ seconds
- Service Impact: None / Minimal

Performance
- Max Throughput: _____ calls/sec
- Average Latency: _____ ms
- Availability: _____%

Cost Analysis
- Hourly Cost: $_____
- Monthly Estimate: $_____
- vs Traditional: ____% savings

Key Learnings
1. _____
2. _____
3. _____

Screenshots
- [ ] Orchestrator Dashboard
- [ ] Service Design
- [ ] Deployment Progress
- [ ] Auto-scaling Event

- [ ] Monitoring Dashboard
- [ ] GitOps Workflow

💡 Key Takeaways

After this lab, you've experienced:

✅ **Orchestration Power**

- Designed complex service visually
- Deployed in minutes, not months
- Zero manual configuration

✅ **Automation Magic**

- Auto-scaling works flawlessly
- Self-healing prevents outages
- Policies enforce intent

✅ **DevOps Reality**

- Infrastructure as Code
- GitOps workflow
- Version-controlled networks

✅ **Business Impact**

- 80% cost reduction
- 99.9% availability
- Instant service delivery

🚀 Next Steps

This Week:

1. **Try Different Services**
   - Deploy vEPC
   - Create vRouter
   - Build vFirewall
2. **Learn YAML/TOSCA**

- Service descriptors
- Policy definitions
- Automation scripts

3. **Explore Open Source**
   - ONAP
   - OSM
   - Cloudify

Career Development:

1. **Get Certified:**
   - NFV certifications
   - Kubernetes (CKA)
   - Cloud providers
2. **Build Portfolio:**
   - GitHub projects
   - Blog about experience
   - Contribute to open source
3. **Network:**
   - Join NFV/SDN groups
   - Attend meetups
   - Follow thought leaders

🤝 Support Resources

Need Help?

- **During Lab:** Raise hand / use chat
- **After Hours:** cloudnative@perfect-tc.com
- **Community:** discord.gg/perfectcloudnative
- **Office Hours:** Thursday 2-4 PM

Documentation:

- Orchestrator Docs: docs.perfect-tc.com/orchestrator

- API Reference: api.perfect-tc.com/docs
- Video Tutorials: youtube.com/perfectcloudnative

🎯 Challenge Yourself!

Advanced Challenge: Multi-Cloud Service

1. Design service spanning:
   - AWS (compute)
   - Azure (database)
   - GCP (AI/ML)
2. Implement cloud bursting
3. Ensure data sovereignty

Innovation Challenge:

Design a cloud-native service for:

- Smart cities in Egypt
- Telehealth for rural areas
- Connected agriculture

Prize: Best design gets featured in newsletter!

🎉 Congratulations!

You've successfully:
✅ Designed a cloud-native service
✅ Deployed with orchestration
✅ Tested automation features
✅ Monitored performance
✅ Experienced GitOps
✅ Calculated cost savings

You're now a Cloud-Native Telecom Engineer!

📸 Social Proof

Don't forget to:

1. Share your success on LinkedIn

2. Tag @PerfectTrainingCenter
3. Use #CloudNativeTelecom
4. Connect with classmates
5. Update your resume!

Perfect Training Center - Transforming Telecom Professionals

Version: 3.0
Updated: November 2024
Next Review: February 2025

🔗 Quick Reference

URLs:

- Main Orchestrator: orchestrator.perfect-tc.com
- Backup: orchestrator2.perfect-tc.com
- Git: git.perfect-tc.com
- Monitoring: monitor.perfect-tc.com

Credentials:

```
Username: student@perfect-tc
Password: CloudNative2024!
```

Commands:

```bash
# Git commands
git add .
git commit -m "message"
git push origin main

# Kubernetes (if needed)
kubectl get pods
kubectl describe service ims
kubectl logs pod-name
```

Ready to Transform Telecom? The Future is Cloud-Native! 🚀