Module 4: Infrastructure Automation

Master Docker, Kubernetes, and cloud deployment for scalable MEV systems

Course Overview

Duration: 170 minutes
Level: Advanced
Prerequisites: Modules 1-3, understanding of cloud services and containerization
Certificate: Available upon completion

Learning Objectives

By the end of this module, you will be able to:
- Design containerized MEV system architectures using Docker
- Implement Kubernetes deployments with auto-scaling and load balancing
- Set up automated CI/CD pipelines for MEV strategy deployment
- Configure cloud infrastructure on AWS, GCP, or Azure
- Implement monitoring, logging, and alerting systems
- Design fault-tolerant and highly available MEV infrastructure

---

Table of Contents

---

1. Container Architecture Design

MEV System Architecture

```
# docker-compose.yml - Complete MEV system architecture
version: '3.8'

services:
  # API Gateway & Load Balancer
  nginx-proxy:
    image: nginx:alpine
    ports:
      - "80:80"
      - "443:443"
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
      - ./nginx/ssl:/etc/nginx/ssl
    depends_on:
      - mev-strategy-api
      - monitoring-api
    networks:
      - mev-network
    restart: unless-stopped

  # MEV Strategy API
  mev-strategy-api:
    build:
      context: ./services/mev-strategy-api
      dockerfile: Dockerfile
    environment:
      - DATABASE_URL=postgresql://mev_user:password@postgres:5432/
mev_db
      - REDIS_URL=redis://redis:6379/0
      - ETHEREUM_RPC_URL=${ETHEREUM_RPC_URL}
      - BSC_RPC_URL=${BSC_RPC_URL}
      - ENVIRONMENT=production
    depends_on:
      - postgres
      - redis
    networks:
      - mev-network
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 1G
```

```
          cpus: '0.5'

  # Blockchain Data Fetcher
  blockchain-fetcher:
    build:
      context: ./services/blockchain-fetcher
      dockerfile: Dockerfile
    environment:
      - ETHEREUM_WS_URL=${ETHEREUM_WS_URL}
      - BSC_WS_URL=${BSC_WS_URL}
      - REDIS_URL=redis://redis:6379/1
      - RATE_LIMIT_REQUESTS_PER_SECOND=100
    depends_on:
      - redis
    networks:
      - mev-network
    restart: unless-stopped
    deploy:
      replicas: 2
      resources:
        limits:
          memory: 512M
          cpus: '0.3'

  # Arbitrage Detector
  arbitrage-detector:
    build:
      context: ./services/arbitrage-detector
      dockerfile: Dockerfile
    environment:
      - DATABASE_URL=postgresql://mev_user:password@postgres:5432/
mev_db
      - REDIS_URL=redis://redis:6379/2
      - MIN_PROFIT_THRESHOLD=1.0
      - MAX_EXECUTION_TIME_MS=50
    depends_on:
      - postgres
      - redis
    networks:
      - mev-network
    restart: unless-stopped
    deploy:
```

```
      replicas: 3
      resources:
        limits:
          memory: 256M
          cpus: '0.2'

  # Liquidation Scanner
  liquidation-scanner:
    build:
      context: ./services/liquidation-scanner
      dockerfile: Dockerfile
    environment:
      - DATABASE_URL=postgresql://mev_user:password@postgres:5432/
mev_db
      - REDIS_URL=redis://redis:6379/3
      - HEALTH_FACTOR_THRESHOLD=1.1
    depends_on:
      - postgres
      - redis
    networks:
      - mev-network
    restart: unless-stopped
    deploy:
      replicas: 2
      resources:
        limits:
          memory: 512M
          cpus: '0.4'

  # Trade Executor
  trade-executor:
    build:
      context: ./services/trade-executor
      dockerfile: Dockerfile
    environment:
      - PRIVATE_KEY=${PRIVATE_KEY}
      - GAS_LIMIT=300000
      - MAX_GAS_PRICE=200000000000
      - WEBHOOK_URL=${WEBHOOK_URL}
    volumes:
      - ./keys:/app/keys:ro
    networks:
```

```
      - mev-network
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 256M
          cpus: '0.2'

  # Database
  postgres:
    image: postgres:15
    environment:
      - POSTGRES_DB=mev_db
      - POSTGRES_USER=mev_user
      - POSTGRES_PASSWORD=password
      - POSTGRES_INITDB_ARGS=--encoding=UTF-8 --lc-collate=C --lc-
ctype=C
    volumes:
      - postgres_data:/var/lib/postgresql/data
      - ./postgres/init:/docker-entrypoint-initdb.d
    networks:
      - mev-network
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 1G
          cpus: '0.5'

  # Redis for caching and pub/sub
  redis:
    image: redis:7-alpine
    command: redis-server --appendonly yes --maxmemory 512mb --
maxmemory-policy allkeys-lru
    volumes:
      - redis_data:/data
    networks:
      - mev-network
    restart: unless-stopped
    deploy:
      resources:
        limits:
```

```
          memory: 512M
          cpus: '0.2'

  # Message Queue
  rabbitmq:
    image: rabbitmq:3-management
    environment:
      - RABBITMQ_DEFAULT_USER=mev_user
      - RABBITMQ_DEFAULT_PASS=password
      - RABBITMQ_DEFAULT_VHOST=mev_vhost
    volumes:
      - rabbitmq_data:/var/lib/rabbitmq
    networks:
      - mev-network
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 512M
          cpus: '0.3'

  # Monitoring Stack
  prometheus:
    image: prom/prometheus:latest
    ports:
      - "9090:9090"
    volumes:
      - ./monitoring/prometheus/prometheus.yml:/etc/prometheus/
prometheus.yml
      - ./monitoring/prometheus/rules:/etc/prometheus/rules
      - prometheus_data:/prometheus
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
      - '--web.console.templates=/etc/prometheus/consoles'
      - '--storage.tsdb.retention.time=200h'
      - '--web.enable-lifecycle'
    networks:
      - mev-network
    restart: unless-stopped
```

```
  grafana:
    image: grafana/grafana:latest
    ports:
      - "3000:3000"
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    volumes:
      - grafana_data:/var/lib/grafana
      - ./monitoring/grafana/dashboards:/var/lib/grafana/dashboards
      - ./monitoring/grafana/provisioning:/etc/grafana/provisioning
    networks:
      - mev-network
    restart: unless-stopped

  # Log Aggregation
  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.11.0
    environment:
      - discovery.type=single-node
      - xpack.security.enabled=false
      - "ES_JAVA_OPTS=-Xms512m -Xmx512m"
    volumes:
      - elasticsearch_data:/usr/share/elasticsearch/data
    networks:
      - mev-network
    restart: unless-stopped
    deploy:
      resources:
        limits:
          memory: 1G
          cpus: '0.5'

  kibana:
    image: docker.elastic.co/kibana/kibana:8.11.0
    ports:
      - "5601:5601"
    environment:
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200
    networks:
      - mev-network
    restart: unless-stopped
```

```
  logstash:
    image: docker.elastic.co/logstash/logstash:8.11.0
    volumes:
      - ./monitoring/logstash/config:/usr/share/logstash/pipeline
      - ./logs:/var/log/mev:ro
    networks:
      - mev-network
    restart: unless-stopped

volumes:
  postgres_data:
  redis_data:
  rabbitmq_data:
  prometheus_data:
  grafana_data:
  elasticsearch_data:

networks:
  mev-network:
    driver: bridge
    ipam:
      config:
        - subnet: 172.20.0.0/16
```

Microservices Architecture Design

```
# microservices/architecture.py
from abc import ABC, abstractmethod
from typing import Dict, List, Optional, Any
from dataclasses import dataclass, field
from enum import Enum
import asyncio
import logging
from datetime import datetime

class ServiceType(Enum):
    API_GATEWAY = "api_gateway"
    BLOCKCHAIN_FETCHER = "blockchain_fetcher"
    OPPORTUNITY_DETECTOR = "opportunity_detector"
    TRADE_EXECUTOR = "trade_executor"
    DATA_PROCESSOR = "data_processor"
    MONITORING = "monitoring"

class DeploymentStrategy(Enum):
    SINGLE_INSTANCE = "single_instance"
    REPLICATED = "replicated"
    LOAD_BALANCED = "load_balanced"
    STATEFUL_SET = "stateful_set"

@dataclass
class ServiceConfig:
    """Configuration for a microservice"""
    name: str
    service_type: ServiceType
    image: str
    version: str
    port: int
    health_check_path: str = "/health"
    replicas: int = 1
    deployment_strategy: DeploymentStrategy =
DeploymentStrategy.SINGLE_INSTANCE

    # Resource requirements
    cpu_request: str = "100m"
    cpu_limit: str = "500m"
    memory_request: str = "128Mi"
    memory_limit: str = "512Mi"
```

```
    # Environment variables
    environment: Dict[str, str] = field(default_factory=dict)

    # Dependencies
    depends_on: List[str] = field(default_factory=list)

    # Configuration
    config_maps: List[str] = field(default_factory=list)
    secrets: List[str] = field(default_factory=list)
    volumes: List[Dict[str, Any]] = field(default_factory=list)

@dataclass
class InfrastructureConfig:
    """Overall infrastructure configuration"""
    cluster_name: str
    namespace: str
    environment: str  # dev, staging, prod
    region: str

    # Services
    services: List[ServiceConfig] = field(default_factory=list)

    # Network configuration
    ingress_enabled: bool = True
    load_balancer_type: str = "nginx"

    # Monitoring
    prometheus_enabled: bool = True
    grafana_enabled: bool = True
    jaeger_enabled: bool = False

    # Security
    rbac_enabled: bool = True
    network_policies_enabled: bool = True

    # Scaling
    hpa_enabled: bool = True
    vpa_enabled: bool = False

class MicroserviceArchitect:
    """Design and manage microservices architecture"""
```

```
    def __init__(self, config: InfrastructureConfig):
        self.config = config
        self.logger = logging.getLogger(__name__)

        # Service registry
        self.service_registry: Dict[str, ServiceConfig] = {}

        # Deployment templates
        self.deployment_templates = self._load_deployment_templates()

        # Initialize services
        self._initialize_services()

    def _initialize_services(self):
        """Initialize all services from configuration"""

        for service_config in self.config.services:
            self.service_registry[service_config.name] = service_config

        self.logger.info(f"Initialized {len(self.service_registry)}
services")

    def _load_deployment_templates(self) -> Dict[str, str]:
        """Load deployment templates for different service types"""

        return {
            ServiceType.API_GATEWAY: "api_gateway_template.yaml",
            ServiceType.BLOCKCHAIN_FETCHER:
"blockchain_fetcher_template.yaml",
            ServiceType.OPPORTUNITY_DETECTOR:
"opportunity_detector_template.yaml",
            ServiceType.TRADE_EXECUTOR: "trade_executor_template.yaml",
            ServiceType.DATA_PROCESSOR: "data_processor_template.yaml",
            ServiceType.MONITORING: "monitoring_template.yaml"
        }

    def generate_kubernetes_manifests(self) -> Dict[str, str]:
        """Generate Kubernetes manifests for all services"""

        manifests = {}

        for service_name, service_config in
```

```
self.service_registry.items():
            # Generate namespace
            namespace_manifest =
self._generate_namespace_manifest(service_config)
            manifests[f"{service_name}_namespace.yaml"] =
namespace_manifest

            # Generate config maps
            for config_map_name in service_config.config_maps:
                config_map_manifest =
self._generate_config_map_manifest(config_map_name, service_config)
                manifests[f"{service_name}_{config_map_name}.yaml"] =
config_map_manifest

            # Generate secrets
            for secret_name in service_config.secrets:
                secret_manifest =
self._generate_secret_manifest(secret_name, service_config)
                manifests[f"{service_name}_{secret_name}.yaml"] =
secret_manifest

            # Generate service deployment
            deployment_manifest =
self._generate_deployment_manifest(service_config)
            manifests[f"{service_name}_deployment.yaml"] =
deployment_manifest

            # Generate service
            service_manifest =
self._generate_service_manifest(service_config)
            manifests[f"{service_name}_service.yaml"] =
service_manifest

            # Generate ingress if needed
            if self.config.ingress_enabled:
                ingress_manifest =
self._generate_ingress_manifest(service_config)
                manifests[f"{service_name}_ingress.yaml"] =
ingress_manifest

            # Generate HPA if auto-scaling enabled
            if self.config.hpa_enabled:
```

```
                hpa_manifest =
self._generate_hpa_manifest(service_config)
                manifests[f"{service_name}_hpa.yaml"] = hpa_manifest

        # Add infrastructure manifests
        manifests.update(self._generate_infrastructure_manifests())

        return manifests

    def _generate_namespace_manifest(self, service_config:
ServiceConfig) -> str:
        """Generate namespace manifest"""

        return f"""apiVersion: v1
kind: Namespace
metadata:
  name: {self.config.namespace}
  labels:
    environment: {self.config.environment}
    cluster: {self.config.cluster_name}
"""

    def _generate_config_map_manifest(self, config_map_name: str,
service_config: ServiceConfig) -> str:
        """Generate config map manifest"""

        # This would contain actual configuration data
        return f"""apiVersion: v1
kind: ConfigMap
metadata:
  name: {config_map_name}
  namespace: {self.config.namespace}
  labels:
    app: {service_config.name}
    component: config
data:
  config.yaml: |
    # Configuration for {service_config.name}
    service:
      name: {service_config.name}
      port: {service_config.port}
      environment: {self.config.environment}
```

```
"""

    def _generate_secret_manifest(self, secret_name: str,
service_config: ServiceConfig) -> str:
        """Generate secret manifest"""

        # Secrets should be externalized and managed properly
        return f"""apiVersion: v1
kind: Secret
metadata:
  name: {secret_name}
  namespace: {self.config.namespace}
  labels:
    app: {service_config.name}
    component: secret
type: Opaque
data:
  # Secrets should be properly encoded
  api-key: <base64-encoded-secret>
  database-password: <base64-encoded-password>
"""

    def _generate_deployment_manifest(self, service_config:
ServiceConfig) -> str:
        """Generate deployment manifest"""

        # Generate environment variables
        env_vars = []
        for key, value in service_config.environment.items():
            env_vars.append(f"        - name: {key}")
            env_vars.append(f"          value: \"{value}\"")

        env_vars_str = "\n".join(env_vars) if env_vars else "        #
No environment variables"

        # Generate dependencies
        readiness_probe = f"""        readinessProbe:
          httpGet:
            path: {service_config.health_check_path}
            port: {service_config.port}
          initialDelaySeconds: 10
          periodSeconds: 5
```

```
          timeoutSeconds: 3
"""

        liveness_probe = f"""        livenessProbe:
          httpGet:
            path: {service_config.health_check_path}
            port: {service_config.port}
          initialDelaySeconds: 30
          periodSeconds: 10
          timeoutSeconds: 5
"""

        return f"""apiVersion: apps/v1
kind: Deployment
metadata:
  name: {service_config.name}
  namespace: {self.config.namespace}
  labels:
    app: {service_config.name}
    version: {service_config.version}
spec:
  replicas: {service_config.replicas}
  selector:
    matchLabels:
      app: {service_config.name}
  template:
    metadata:
      labels:
        app: {service_config.name}
        version: {service_config.version}
    spec:
      containers:
      - name: {service_config.name}
        image: {service_config.image}:{service_config.version}
        ports:
        - containerPort: {service_config.port}
          name: http
        env:
{env_vars_str}
        resources:
          requests:
            cpu: {service_config.cpu_request}
```

```
            memory: {service_config.memory_request}
          limits:
            cpu: {service_config.cpu_limit}
            memory: {service_config.memory_limit}
{readiness_probe}
{liveness_probe}
      restartPolicy: Always
"""

    def _generate_service_manifest(self, service_config: ServiceConfig)
-> str:
        """Generate service manifest"""

        return f"""apiVersion: v1
kind: Service
metadata:
  name: {service_config.name}
  namespace: {self.config.namespace}
  labels:
    app: {service_config.name}
spec:
  type: ClusterIP
  ports:
  - port: {service_config.port}
    targetPort: {service_config.port}
    protocol: TCP
    name: http
  selector:
    app: {service_config.name}
"""

    def _generate_ingress_manifest(self, service_config: ServiceConfig)
-> str:
        """Generate ingress manifest"""

        if service_config.service_type == ServiceType.API_GATEWAY:
            return f"""apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: {service_config.name}-ingress
  namespace: {self.config.namespace}
  annotations:
```

```
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: api.mev-system.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: {service_config.name}
            port:
              number: {service_config.port}
"""
        else:
            # Internal services don't need ingress
            return ""

    def _generate_hpa_manifest(self, service_config: ServiceConfig) ->
str:
        """Generate horizontal pod autoscaler manifest"""

        return f"""apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: {service_config.name}-hpa
  namespace: {self.config.namespace}
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: {service_config.name}
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
```

```
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
"""

    def _generate_infrastructure_manifests(self) -> Dict[str, str]:
        """Generate infrastructure-level manifests"""

        manifests = {}

        # RBAC
        if self.config.rbac_enabled:
            manifests["rbac.yaml"] = self._generate_rbac_manifest()

        # Network policies
        if self.config.network_policies_enabled:
            manifests["network-policies.yaml"] =
self._generate_network_policies()

        # Monitoring
        if self.config.prometheus_enabled:
            manifests["monitoring.yaml"] =
self._generate_monitoring_manifest()

        return manifests

    def _generate_rbac_manifest(self) -> str:
        """Generate RBAC manifests"""

        return f"""apiVersion: v1
kind: ServiceAccount
metadata:
  name: mev-system-sa
  namespace: {self.config.namespace}
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: mev-system-role
```

```
rules:
- apiGroups: [""]
  resources: ["pods", "services", "configmaps", "secrets"]
  verbs: ["get", "list", "watch", "create", "update", "patch",
"delete"]
- apiGroups: ["apps"]
  resources: ["deployments", "replicasets"]
  verbs: ["get", "list", "watch", "create", "update", "patch",
"delete"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: mev-system-rolebinding
roleRef:
  apiGroup: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  name: mev-system-role
subjects:
- kind: ServiceAccount
  name: mev-system-sa
  namespace: {self.config.namespace}
"""

    def _generate_network_policies(self) -> str:
        """Generate network policies"""

        return f"""apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: {self.config.namespace}
spec:
  podSelector: {{}}
  policyTypes:
  - Ingress
  - Egress
---
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-api-gateway
```

```
  namespace: {self.config.namespace}
spec:
  podSelector:
    matchLabels:
      app: nginx-proxy
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: ingress-nginx
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: mev-strategy-api
    ports:
    - protocol: TCP
      port: 8080
"""

    def _generate_monitoring_manifest(self) -> str:
        """Generate monitoring setup manifest"""

        return """apiVersion: v1
kind: ServiceAccount
metadata:
  name: prometheus
  namespace: monitoring
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: prometheus
rules:
- apiGroups: [""]
  resources: ["nodes", "nodes/proxy", "services", "endpoints", "pods"]
  verbs: ["get", "list", "watch"]
- apiGroups: ["extensions"]
  resources: ["ingresses"]
```

```
  verbs: ["get", "list", "watch"]
- nonResourceURLs: ["/metrics"]
  verbs: ["get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: prometheus
roleRef:
  apiGroup: rbac.authorization.k8s.io/v1
  kind: ClusterRole
  name: prometheus
subjects:
- kind: ServiceAccount
  name: prometheus
  namespace: monitoring
"""
```

---

2. Docker Implementation

Dockerfile Best Practices

```
# services/mev-strategy-api/Dockerfile
# Multi-stage build for optimal image size

# Build stage
FROM golang:1.21-alpine AS builder

# Install dependencies
RUN apk add --no-cache git ca-certificates tzdata

# Set working directory
WORKDIR /app

# Copy go mod files
COPY go.mod go.sum ./

# Download dependencies
RUN go mod download

# Copy source code
COPY . .

# Build the application
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix cgo -o main .

# Final stage
FROM alpine:latest

# Install ca-certificates for HTTPS calls
RUN apk --no-cache add ca-certificates

# Create non-root user
RUN addgroup -g 1001 -S mevgroup && \
    adduser -u 1001 -S mevuser -G mevgroup

# Set working directory
WORKDIR /app

# Copy binary from builder stage
COPY --from=builder /app/main .

# Copy configuration files
COPY config/ ./config/
```

```
# Set permissions
RUN chown -R mevuser:mevgroup /app && \
    chmod +x main

# Switch to non-root user
USER mevuser

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=60s --
retries=3 \
    CMD wget --no-verbose --tries=1 --spider http://localhost:8080/
health || exit 1

# Expose port
EXPOSE 8080

# Run the application
CMD ["./main"]
```

```
# services/blockchain-fetcher/Dockerfile
# Python-based service with optimized dependencies

FROM python:3.11-slim as builder

# Install system dependencies
RUN apt-get update && apt-get install -y \
    gcc \
    g++ \
    make \
    libffi-dev \
    libssl-dev \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Set working directory
WORKDIR /app

# Copy requirements first for better caching
COPY requirements.txt .

# Install Python dependencies
RUN pip install --no-cache-dir --user -r requirements.txt

# Production stage
FROM python:3.11-slim as production

# Install runtime dependencies
RUN apt-get update && apt-get install -y \
    curl \
    && rm -rf /var/lib/apt/lists/*

# Create non-root user
RUN groupadd -r mevuser && useradd -r -g mevuser mevuser

# Set working directory
WORKDIR /app

# Copy installed packages from builder
COPY --from=builder /root/.local /home/mevuser/.local

# Copy application code
```

```
COPY --chown=mevuser:mevuser . .

# Set PATH to include user binaries
ENV PATH=/home/mevuser/.local/bin:$PATH

# Health check
HEALTHCHECK --interval=30s --timeout=10s --start-period=30s --
retries=3 \
    CMD curl -f http://localhost:8081/health || exit 1

# Expose port
EXPOSE 8081

# Switch to non-root user
USER mevuser

# Run the application
CMD ["python", "main.py"]
```

Docker Compose for Development

```
# docker-compose.dev.yml
version: '3.8'

services:
  # Development API with hot reload
  mev-strategy-api-dev:
    build:
      context: ./services/mev-strategy-api
      dockerfile: Dockerfile.dev
    environment:
      - ENVIRONMENT=development
      - LOG_LEVEL=debug
      - DATABASE_URL=postgresql://mev_dev:dev_password@postgres-dev:
5432/mev_dev_db
      - REDIS_URL=redis://redis-dev:6379/0
    volumes:
      - ./services/mev-strategy-api:/app
      - /app/go/pkg
    ports:
      - "8080:8080"
    depends_on:
      - postgres-dev
      - redis-dev
    networks:
      - dev-network
    profiles:
      - dev

  # Development database with sample data
  postgres-dev:
    image: postgres:15
    environment:
      - POSTGRES_DB=mev_dev_db
      - POSTGRES_USER=mev_dev
      - POSTGRES_PASSWORD=dev_password
    volumes:
      - postgres_dev_data:/var/lib/postgresql/data
      - ./dev-setup/sql:/docker-entrypoint-initdb.d
    ports:
      - "5432:5432"
    networks:
      - dev-network
```

```
    profiles:
      - dev

  # Redis for development
  redis-dev:
    image: redis:7-alpine
    command: redis-server --appendonly yes --save 60 1 --loglevel
warning
    volumes:
      - redis_dev_data:/data
    ports:
      - "6379:6379"
    networks:
      - dev-network
    profiles:
      - dev

  # Adminer for database management
  adminer:
    image: adminer
    restart: always
    ports:
      - "8082:8080"
    environment:
      ADMINER_DEFAULT_SERVER: postgres-dev
    depends_on:
      - postgres-dev
    networks:
      - dev-network
    profiles:
      - dev

  # LocalStack for AWS services emulation
  localstack:
    image: localstack/localstack:latest
    ports:
      - "4566:4566"
    environment:
      - SERVICES=s3,secretsmanager,sts
      - DEBUG=1
      - DATA_DIR=/tmp/localstack/data
    volumes:
```

```
      - "/tmp/localstack:/tmp/localstack"
      - "/var/run/docker.sock:/var/run/docker.sock"
    networks:
      - dev-network
    profiles:
      - dev

  # Jaeger for distributed tracing
  jaeger:
    image: jaegertracing/all-in-one:latest
    ports:
      - "16686:16686"
      - "14268:14268"
    environment:
      - COLLECTOR_OTLP_ENABLED=true
    networks:
      - dev-network
    profiles:
      - dev

  # Mailhog for email testing
  mailhog:
    image: mailhog/mailhog
    ports:
      - "1025:1025"
      - "8025:8025"
    networks:
      - dev-network
    profiles:
      - dev

volumes:
  postgres_dev_data:
  redis_dev_data:

networks:
  dev-network:
    driver: bridge
```

Environment-Specific Configurations

```yaml
# config/environments/production.yml
# Production environment configuration

server:
  port: 8080
  read_timeout: 30s
  write_timeout: 30s
  idle_timeout: 120s
  max_header_bytes: 1048576

database:
  url: ${DATABASE_URL}
  max_open_connections: 25
  max_idle_connections: 25
  connection_max_lifetime: 300s
  connection_max_idle_time: 60s

redis:
  url: ${REDIS_URL}
  pool_size: 10
  max_retries: 3
  retry_delay: 100ms
  key_prefix: "mev:prod:"

blockchain:
  ethereum:
    rpc_url: ${ETHEREUM_RPC_URL}
    ws_url: ${ETHEREUM_WS_URL}
    chain_id: 1
    confirmations: 12
    gas_limit_multiplier: 1.2

  bsc:
    rpc_url: ${BSC_RPC_URL}
    ws_url: ${BSC_WS_URL}
    chain_id: 56
    confirmations: 12
    gas_limit_multiplier: 1.1

mev:
  strategies:
    arbitrage:
```

```
      enabled: true
      min_profit_usd: 5.0
      max_execution_time_ms: 50
      gas_price_limit_gwei: 150
      slippage_tolerance: 0.005

  liquidation:
    enabled: true
      health_factor_threshold: 1.1
      max_gas_price_gwei: 200
      min_bonus_percentage: 0.05

  monitoring:
    enabled: true
    check_interval_seconds: 1
    alerting:
      enabled: true
      webhook_url: ${ALERT_WEBHOOK_URL}
      min_profit_threshold: 10.0

trading:
  private_key: ${PRIVATE_KEY}
  max_gas_price_gwei: 200
  gas_limit: 300000
  nonce_management: true
  transaction_timeout_seconds: 300
  max_retry_attempts: 3

monitoring:
  prometheus:
    enabled: true
    metrics_port: 9090

  logging:
    level: info
    format: json
    output: stdout

  tracing:
    enabled: true
    jaeger_endpoint: ${JAEGER_ENDPOINT}
```

```
security:
  cors_allowed_origins: ${CORS_ALLOWED_ORIGINS}
  rate_limiting:
    enabled: true
    requests_per_minute: 1000
    burst_size: 100

scaling:
  auto_scaling:
    enabled: true
    min_replicas: 2
    max_replicas: 10
    cpu_threshold: 70
    memory_threshold: 80
    scale_up_cooldown: 300s
    scale_down_cooldown: 600s
```

---

3. Kubernetes Orchestration

Kubernetes Deployment Files

```yaml
# k8s/mev-strategy-api-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mev-strategy-api
  namespace: mev-system
  labels:
    app: mev-strategy-api
    version: v1.0.0
    component: api
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: mev-strategy-api
  template:
    metadata:
      labels:
        app: mev-strategy-api
        version: v1.0.0
      annotations:
        prometheus.io/scrape: "true"
        prometheus.io/port: "9090"
        prometheus.io/path: "/metrics"
    spec:
      serviceAccountName: mev-system-sa
      containers:
      - name: mev-strategy-api
        image: mev-system/mev-strategy-api:v1.0.0
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 8080
          name: http
          protocol: TCP
        - containerPort: 9090
          name: metrics
          protocol: TCP
```

```
        env:
        - name: ENVIRONMENT
          value: "production"
        - name: DATABASE_URL
          valueFrom:
            secretKeyRef:
              name: database-secret
              key: url
        - name: REDIS_URL
          valueFrom:
            configMapKeyRef:
              name: redis-config
              key: url
        - name: ETHEREUM_RPC_URL
          valueFrom:
            secretKeyRef:
              name: blockchain-secrets
              key: ethereum-rpc-url
        - name: PRIVATE_KEY
          valueFrom:
            secretKeyRef:
              name: trading-secrets
              key: private-key
        - name: LOG_LEVEL
          value: "info"
        resources:
          requests:
            cpu: 500m
            memory: 512Mi
          limits:
            cpu: 1000m
            memory: 1Gi
        livenessProbe:
          httpGet:
            path: /health
            port: 8080
          initialDelaySeconds: 60
          periodSeconds: 30
          timeoutSeconds: 10
          failureThreshold: 3
        readinessProbe:
          httpGet:
```

```
        path: /ready
        port: 8080
      initialDelaySeconds: 30
      periodSeconds: 10
      timeoutSeconds: 5
      failureThreshold: 3
    volumeMounts:
    - name: config-volume
      mountPath: /app/config
      readOnly: true
    - name: logs-volume
      mountPath: /app/logs
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      runAsNonRoot: true
      runAsUser: 1001
      capabilities:
        drop:
        - ALL
  volumes:
  - name: config-volume
    configMap:
      name: mev-config
  - name: logs-volume
    emptyDir: {}
  securityContext:
    fsGroup: 1001
  restartPolicy: Always
  terminationGracePeriodSeconds: 30
  nodeSelector:
    node-type: general
  tolerations:
  - key: "mev-system"
    operator: "Equal"
    value: "enabled"
    effect: "NoSchedule"
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
```

```
              labelSelector:
                matchExpressions:
                - key: app
                  operator: In
                  values:
                  - mev-strategy-api
              topologyKey: kubernetes.io/hostname
```

```yaml
# k8s/hpa-mev-strategy-api.yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: mev-strategy-api-hpa
  namespace: mev-system
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: mev-strategy-api
  minReplicas: 3
  maxReplicas: 20
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 70
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
  - type: Pods
    pods:
      metric:
        name: mev_opportunities_per_second
      target:
        type: AverageValue
        averageValue: "10"
  behavior:
    scaleDown:
      stabilizationWindowSeconds: 300
      policies:
      - type: Percent
        value: 10
        periodSeconds: 60
    scaleUp:
      stabilizationWindowSeconds: 60
```

```
      policies:
      - type: Percent
        value: 50
        periodSeconds: 60
      - type: Pods
        value: 2
        periodSeconds: 60
```

```
# k8s/pdb-mev-strategy-api.yaml
apiVersion: policy/v1
kind: PodDisruptionBudget
metadata:
  name: mev-strategy-api-pdb
  namespace: mev-system
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: mev-strategy-api
```

Service Mesh Configuration

```yaml
# k8s/istio-gateway.yaml
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: mev-system-gateway
  namespace: istio-system
spec:
  selector:
    istio: ingressgateway
  servers:
  - port:
      number: 80
      name: http
      protocol: HTTP
    hosts:
    - api.mev-system.com
    - grafana.mev-system.com
    - prometheus.mev-system.com
  - port:
      number: 443
      name: https
      protocol: HTTPS
    tls:
      mode: SIMPLE
      credentialName: mev-system-tls
    hosts:
    - api.mev-system.com
    - grafana.mev-system.com
    - prometheus.mev-system.com
---
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: mev-system-vs
  namespace: mev-system
spec:
  hosts:
  - api.mev-system.com
  gateways:
  - mev-system-gateway
  http:
  - match:
```

```
    - uri:
        prefix: /api/v1/
    route:
    - destination:
        host: mev-strategy-api
        port:
          number: 8080
    timeout: 30s
    retries:
      attempts: 3
      perTryTimeout: 10s
  - match:
    - uri:
        prefix: /metrics
    route:
    - destination:
        host: mev-strategy-api
        port:
          number: 9090
  - match:
    - uri:
        prefix: /health
    route:
    - destination:
        host: mev-strategy-api
        port:
          number: 8080
    timeout: 5s
---
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: mev-strategy-api-dr
  namespace: mev-system
spec:
  host: mev-strategy-api
  trafficPolicy:
    loadBalancer:
      consistentHash:
        httpHeaderName: "X-User-ID"
    connectionPool:
      tcp:
```

```
      maxConnections: 100
    http:
      http1MaxPendingRequests: 50
      maxRequestsPerConnection: 10
  circuitBreaker:
    consecutiveErrors: 3
    interval: 30s
    baseEjectionTime: 30s
    maxEjectionPercent: 50
portLevelSettings:
- port:
    number: 8080
  loadBalancer:
    simple: LEAST_CONN
- port:
    number: 9090
  loadBalancer:
    simple: ROUND_ROBIN
```

---

4. Cloud Deployment

AWS EKS Deployment

```
# aws/eks-cluster.yaml
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: mev-system-prod
  region: us-west-2
  version: "1.28"

nodeGroups:
  - name: general-workers
    instanceType: m5.xlarge
    desiredCapacity: 3
    minSize: 2
    maxSize: 10
    volumeSize: 50
    amiFamily: AmazonLinux2
    ssh:
      allow: true
      publicKeyName: mev-system-key
    tags:
      nodegroup-role: worker
      kubernetes.io/cluster/mev-system-prod: owned
    iam:
      attachPolicyARNs:
        - arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy
        - arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy
        - arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly
        - arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy
    kubeletExtraConfig:
      kubeReserved:
        cpu: "200m"
        memory: "200Mi"
      systemReserved:
        cpu: "100m"
        memory: "100Mi"
      evictionHard:
        memory.available: "200Mi"

  - name: compute-optimized
    instanceType: c5.2xlarge
    desiredCapacity: 2
```

```
    minSize: 1
    maxSize: 8
    volumeSize: 30
    amiFamily: AmazonLinux2
    taints:
      - key: "compute-optimized"
        value: "true"
        effect: NO_SCHEDULE
    labels:
      workload-type: compute-intensive
    tags:
      nodegroup-role: compute-worker

addons:
  - name: vpc-cni
  - name: coredns
  - name: kube-proxy
  - name: aws-ebs-csi-driver
  - name: aws-load-balancer-controller
  - name: metrics-server
  - name: cluster-autoscaler

cloudWatch:
  clusterLogging:
    enableTypes: ["*"]
```

```
#!/bin/bash
# aws/deploy-eks.sh

set -e

# Configuration
CLUSTER_NAME="mev-system-prod"
REGION="us-west-2"
NAMESPACE="mev-system"

echo "🚀 Deploying MEV system to AWS EKS..."

# 1. Create EKS cluster
echo "📦 Creating EKS cluster..."
eksctl create cluster \
  --name $CLUSTER_NAME \
  --region $REGION \
  --version 1.28 \
  --nodegroup-name general-workers \
  --node-type m5.xlarge \
  --nodes 3 \
  --nodes-min 2 \
  --nodes-max 10 \
  --managed

# 2. Create namespace
echo "🏗️ Creating namespace..."
kubectl create namespace $NAMESPACE --dry-run=client -o yaml | kubectl
apply -f -

# 3. Install EBS CSI driver
echo "💾 Installing EBS CSI driver..."
eksctl create addon \
  --name aws-ebs-csi-driver \
  --cluster $CLUSTER_NAME \
  --region $REGION \
  --force

# 4. Install AWS Load Balancer Controller
echo "⚖️ Installing AWS Load Balancer Controller..."
kubectl apply -k "github.com/aws/eks-charts/stable/aws-load-balancer-
controller//crds?ref=master"
```

```
helm repo add eks https://aws.github.io/eks-charts
helm repo update

helm install aws-load-balancer-controller eks/aws-load-balancer-
controller \
  --set clusterName=$CLUSTER_NAME \
  --set serviceAccount.create=false \
  --set serviceAccount.name=aws-load-balancer-controller \
  -n kube-system

# 5. Install metrics server
echo "📊 Installing metrics server..."
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/
releases/latest/download/components.yaml

# 6. Install cluster autoscaler
echo "🔄 Installing cluster autoscaler..."
kubectl apply -f https://raw.githubusercontent.com/kubernetes/
autoscaler/master/cluster-autoscaler/cloudprovider/aws/examples/
cluster-autoscaler-autodiscover.yaml

kubectl annotate deployment.apps/cluster-autoscaler \
  -n kube-system \
  cluster-autoscaler.kubernetes.io/safe-to-evict="false"

# 7. Install NGINX Ingress
echo "🚪 Installing NGINX Ingress..."
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update

helm install ingress-nginx ingress-nginx/ingress-nginx \
  --set controller.service.externalTrafficPolicy=Local \
  -n ingress-nginx \
  --create-namespace

# 8. Install Cert Manager
echo "🔐 Installing Cert Manager..."
kubectl apply -f https://github.com/cert-manager/cert-manager/releases/
download/v1.13.0/cert-manager.yaml

# 9. Install Prometheus and Grafana
```

```
echo "📈 Installing monitoring stack..."
helm repo add prometheus-community https://prometheus-
community.github.io/helm-charts
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update

helm install prometheus prometheus-community/kube-prometheus-stack \
  --namespace monitoring \
  --create-namespace \
  -f aws/prometheus-values.yaml

# 10. Install Istio (optional)
echo "🕸️ Installing Istio service mesh..."
istioctl install --set values.global.meshID=mev-system-mesh \
  --set values.global.cluster=mev-system-prod \
  --set values.global.network=mev-system-network

# 11. Deploy MEV system
echo "🎯 Deploying MEV system..."
kubectl apply -f aws/mev-system-namespace.yaml
kubectl apply -f aws/secrets/
kubectl apply -f aws/configmaps/
kubectl apply -f aws/ingress/
kubectl apply -f aws/certificates/

# Deploy all services
kubectl apply -f k8s/

echo "✅ Deployment complete!"
echo "📋 Next steps:"
echo "   - Configure DNS records for *.mev-system.com"
echo "   - Set up monitoring alerts"
echo "   - Run system tests"
echo "   - Enable backup strategies"
```

Terraform Infrastructure as Code

```
# aws/main.tf
terraform {
  required_version = ">= 1.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "~> 2.20"
    }
    helm = {
      source  = "hashicorp/helm"
      version = "~> 2.10"
    }
  }

  backend "s3" {
    bucket = "mev-system-terraform-state"
    key    = "infrastructure/terraform.tfstate"
    region = "us-west-2"
    encrypt = true
  }
}

provider "aws" {
  region = var.aws_region

  default_tags {
    tags = {
      Project     = "MEV-System"
      Environment = var.environment
      ManagedBy   = "Terraform"
      CostCenter  = "MEV-Operations"
    }
  }
}

provider "kubernetes" {
  host                   = data.aws_eks_cluster.cluster.endpoint
  cluster_ca_certificate =
```

```
base64decode(data.aws_eks_cluster.cluster.certificate_authority[0].data)
  token                  = data.aws_eks_cluster_auth.cluster.token
}

provider "helm" {
  kubernetes {
    host                   = data.aws_eks_cluster.cluster.endpoint
    cluster_ca_certificate =
base64decode(data.aws_eks_cluster.cluster.certificate_authority[0].data)
    token                  = data.aws_eks_cluster_auth.cluster.token
  }
}

# Variables
variable "aws_region" {
  description = "AWS region"
  type        = string
  default     = "us-west-2"
}

variable "environment" {
  description = "Environment name"
  type        = string
  default     = "production"
}

variable "cluster_name" {
  description = "EKS cluster name"
  type        = string
  default     = "mev-system"
}

variable "node_groups" {
  description = "EKS node group configurations"
  type = map(object({
    instance_types = list(string)
    capacity_type  = string
    scaling_config = object({
      desired_size = number
      max_size     = number
      min_size     = number
    })
```

```
    disk_size = number
    tags      = map(string)
  }))
}

# Data sources
data "aws_availability_zones" "available" {
  state = "available"
}

data "aws_eks_cluster" "cluster" {
  name = var.cluster_name
}

data "aws_eks_cluster_auth" "cluster" {
  name = var.cluster_name
}

# VPC Configuration
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  version = "~> 5.0"

  name = "${var.cluster_name}-vpc"
  cidr = "10.0.0.0/16"

  azs             = data.aws_availability_zones.available.names
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24", "10.0.103.0/24"]

  enable_nat_gateway = true
  single_nat_gateway = true
  enable_vpn_gateway = true
  enable_dns_hostnames = true
  enable_dns_support = true

  public_subnet_tags = {
    "kubernetes.io/role/elb" = "1"
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }

  private_subnet_tags = {
```

```
    "kubernetes.io/role/internal-elb" = "1"
    "kubernetes.io/cluster/${var.cluster_name}" = "shared"
  }

  tags = {
    Name = "${var.cluster_name}-vpc"
  }
}

# EKS Cluster
module "eks" {
  source = "terraform-aws-modules/eks/aws"
  version = "~> 19.15"

  cluster_name    = var.cluster_name
  cluster_version = "1.28"

  vpc_id                         = module.vpc.vpc_id
  subnet_ids                     = module.vpc.private_subnets
  cluster_endpoint_public_access = true
  cluster_endpoint_private_access = true

  # Cluster logging
  cluster_enabled_log_types = ["api", "audit", "authenticator",
"controllerManager", "scheduler"]

  # IRSA (IAM Roles for Service Accounts)
  enable_irsa = true

  # Add-ons
  cluster_addons = {
    coredns = {
      most_recent = true
    }
    kube-proxy = {
      most_recent = true
    }
    vpc-cni = {
      most_recent = true
    }
  }
```

```
# Node groups
node_group_defaults = {
  ami_type       = "AL2_x86_64"
  capacity_type  = "ON_DEMAND"
  instance_types = ["m5.large"]

  # Attach additional IAM policies to the worker node groups
  attach_iam_policy_arns = [
    "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy",
    "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy",
    "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly",
  ]

  # Taints to limit which pods can schedule on these nodes
  taints = []

  # Additional security groups to attach to the node group
  additional_security_group_ids = []

  # Key name to associate with the node group instances
  key_name = ""

  # Maximum pods per node
  max_pods_per_node = 110

  update_config = {
    max_unavailable_percentage = 25
  }

  # Custom AMIs
  custom_ami_id = ""

  # Encryption for node group resources
  encrypt_boot_volume = true

  # Monitoring
  enable_monitoring = true

  # Placement group
  placement = {
    group_strategy = "spread"
  }
```

```
  # Public access to nodes
  public_access = true

  # SSH access
  ssh = {
    allow = true
    public_key_name = "mev-system-key"
    source_security_groups = []
    iam_instance_profile_name = null
  }

  # Subnets to use for the node group
  subnet_ids = []

  # Tags to apply to the node group
  tags = {
    Environment = var.environment
    Project     = "MEV-System"
  }

  # Additional volumes to attach to the nodes
  additional_volumes = [
    {
      type      = "gp3"
      size      = 50
      iops      = 3000
      throughput = 150
      encrypted = true
    }
  ]
}

# Custom node groups
node_groups = var.node_groups

tags = {
  Environment = var.environment
  Project     = "MEV-System"
}

manage_aws_auth_configmap = true
```

```
  aws_auth_roles = [
    {
      rolearn  = aws_iam_role.eks_admin_role.arn
      username = "eks-admin"
      groups   = ["system:masters"]
    },
  ]
}

# IAM Roles and Policies
resource "aws_iam_role" "eks_admin_role" {
  name = "${var.cluster_name}-eks-admin"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          AWS = aws_iam_user.eks_admin_user.arn
        }
      }
    ]
  })
}

resource "aws_iam_user" "eks_admin_user" {
  name = "${var.cluster_name}-eks-admin"
}

resource "aws_iam_user_policy_attachment" "eks_admin_user_attachment" {
  user       = aws_iam_user.eks_admin_user.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
}

# RDS Database
resource "aws_db_subnet_group" "mev_db_subnet_group" {
  name       = "${var.cluster_name}-db-subnet-group"
  subnet_ids = module.vpc.private_subnets

  tags = {
```

```
    Name = "${var.cluster_name}-db-subnet-group"
  }
}

resource "aws_security_group" "rds_sg" {
  name_prefix = "${var.cluster_name}-rds-"
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port       = 5432
    to_port         = 5432
    protocol        = "tcp"
    security_groups = [module.eks.node_security_group_id]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "${var.cluster_name}-rds-sg"
  }
}

resource "aws_db_instance" "mev_db" {
  identifier = "${var.cluster_name}-db"

  engine         = "postgres"
  engine_version = "15.4"
  instance_class = "db.t3.medium"

  allocated_storage     = 100
  max_allocated_storage = 1000
  storage_type          = "gp3"
  storage_encrypted     = true

  db_name  = "mevdb"
  username = "mevadmin"
  password = random_password.db_password.result
```

```
  vpc_security_group_ids = [aws_security_group.rds_sg.id]
  db_subnet_group_name   = aws_db_subnet_group.mev_db_subnet_group.name

  backup_retention_period = 30
  backup_window          = "03:00-04:00"
  maintenance_window     = "sun:04:00-sun:05:00"

  skip_final_snapshot = false
  final_snapshot_identifier = "<span class="math-inline"
style="display: inline;"><math xmlns="http://www.w3.org/1998/Math/
MathML" display="inline"><mrow><mrow><mi>v</mi><mi>a</mi><mi>r</
mi><mo>&#x0002E;</mo><mi>c</mi><mi>l</mi><mi>u</mi><mi>s</mi><mi>t</
mi><mi>e</mi><msub><mi>r</mi><mi>n</mi></msub><mi>a</mi><mi>m</
mi><mi>e</mi></mrow><mo>&#x02212;</mo><mi>f</mi><mi>i</mi><mi>n</
mi><mi>a</mi><mi>l</mi><mo>&#x02212;</mo><mi>s</mi><mi>n</mi><mi>a</
mi><mi>p</mi><mi>s</mi><mi>h</mi><mi>o</mi><mi>t</mi><mo>&#x02212;</
mo></mrow></math></span>{formatdate("YYYY-MM-DD-hhmm", timestamp())}"

  performance_insights_enabled = true
  monitoring_interval         = 60
  monitoring_role_arn         =
aws_iam_role.rds_enhanced_monitoring_role.arn

  tags = {
    Name = "${var.cluster_name}-db"
  }

  deletion_protection = true
}

resource "random_password" "db_password" {
  length  = 32
  special = true
}

resource "aws_ssm_parameter" "db_password" {
  name        = "/${var.cluster_name}/database/password"
  description = "Database password for MEV system"
  type        = "SecureString"
  value       = random_password.db_password.result
```

```
  tags = {
    Environment = var.environment
    Project     = "MEV-System"
  }
}

resource "aws_iam_role" "rds_enhanced_monitoring_role" {
  name = "${var.cluster_name}-rds-monitoring-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "monitoring.rds.amazonaws.com"
        }
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment"
"rds_enhanced_monitoring_role_attachment" {
  role       = aws_iam_role.rds_enhanced_monitoring_role.name
  policy_arn = "arn:aws:iam::aws:policy/service-role/
AmazonRDSEnhancedMonitoringRole"
}

# ElastiCache Redis
resource "aws_elasticache_subnet_group" "mev_redis_subnet_group" {
  name       = "${var.cluster_name}-redis-subnet-group"
  subnet_ids = module.vpc.private_subnets

  tags = {
    Name = "${var.cluster_name}-redis-subnet-group"
  }
}

resource "aws_security_group" "redis_sg" {
  name_prefix = "${var.cluster_name}-redis-"
```

```
  vpc_id      = module.vpc.vpc_id

  ingress {
    from_port       = 6379
    to_port         = 6379
    protocol        = "tcp"
    security_groups = [module.eks.node_security_group_id]
  }

  tags = {
    Name = "${var.cluster_name}-redis-sg"
  }
}

resource "aws_elasticache_replication_group" "mev_redis" {
  replication_group_id         = "${var.cluster_name}-redis"
  description                  = "Redis cluster for MEV system"

  node_type                    = "cache.r6g.large"
  port                         = 6379
  parameter_group_name         =
aws_elasticache_parameter_group.redis.name
  num_cache_clusters           = 3

  subnet_group_name            =
aws_elasticache_subnet_group.mev_redis_subnet_group.name
  security_group_ids           = [aws_security_group.redis_sg.id]

  at_rest_encryption_enabled   = true
  transit_encryption_enabled   = true
  auth_token                   =
random_password.redis_auth_token.result
  kms_key_id                   = aws_kms_key.redis_key.arn

  backup_retention_period      = 7
  snapshot_window             = "03:00-05:00"

  log_delivery_configuration {
    destination      = aws_cloudwatch_log_group.redis_slow.name
    destination_type = "cloudwatch-logs"
    log_format       = "text"
    log_type         = "slow-log"
```

```
  }

  tags = {
    Name = "${var.cluster_name}-redis"
  }
}

resource "random_password" "redis_auth_token" {
  length  = 32
  special = true
}

resource "aws_kms_key" "redis_key" {
  description             = "KMS key for Redis encryption"
  deletion_window_in_days = 30
  enable_key_rotation     = true

  tags = {
    Name = "${var.cluster_name}-redis-key"
  }
}

# CloudWatch Monitoring
resource "aws_cloudwatch_log_group" "mev_system_logs" {
  name              = "/aws/eks/${var.cluster_name}/mev-system"
  retention_in_days = 30
  kms_key_id        = aws_kms_key.cloudwatch_logs_key.arn
}

resource "aws_kms_key" "cloudwatch_logs_key" {
  description             = "KMS key for CloudWatch Logs encryption"
  deletion_window_in_days = 30
  enable_key_rotation     = true
}

# Alerting
resource "aws_sns_topic" "mev_system_alerts" {
  name = "${var.cluster_name}-alerts"
}

# Outputs
output "cluster_endpoint" {
```

```
  description = "Endpoint for EKS control plane"
  value       = data.aws_eks_cluster.cluster.endpoint
}

output "cluster_security_group_id" {
  description = "Security group ids attached to the cluster control
plane"
  value       = module.eks.cluster_security_group_id
}

output "cluster_iam_role_name" {
  description = "IAM role name associated with EKS cluster"
  value       = module.eks.cluster_iam_role_name
}

output "database_endpoint" {
  description = "RDS instance endpoint"
  value       = aws_db_instance.mev_db.endpoint
}

output "redis_endpoint" {
  description = "Redis cluster endpoint"
  value       =
aws_elasticache_replication_group.mev_redis.primary_endpoint_address
}
```

Summary

This comprehensive module on infrastructure automation provides essential skills for deploying and managing production MEV systems. Key achievements:

✅ What You Learned:

- **Container architecture design** with Docker and Docker Compose for local development
- **Kubernetes orchestration** with deployments, services, ingress, and auto-scaling
- **Cloud deployment strategies** using AWS EKS with Terraform infrastructure as code
- **CI/CD pipeline implementation** with automated testing, building, and deployment
- **Monitoring and observability** with Prometheus, Grafana, and distributed tracing
- **Security and compliance** with RBAC, network policies, and secret management

🚀 Production Implementation:

- **Multi-environment deployment** with development, staging, and production configurations
- **Auto-scaling infrastructure** based on CPU, memory, and custom metrics
- **High availability setup** with multi-AZ deployment and load balancing
- **Disaster recovery** with automated backups and failover mechanisms
- **Performance optimization** with resource limits, pod anti-affinity, and quality of service

💡 Best Practices:

- **Infrastructure as Code** with version-controlled configurations
- **Blue-green deployments** for zero-downtime updates
- **Service mesh integration** for advanced traffic management
- **Secret management** with proper encryption and rotation
- **Monitoring-first approach** with comprehensive alerting and observability

The infrastructure automation skills from this module enable building robust, scalable, and maintainable MEV systems that can operate reliably in production environments with enterprise-grade reliability and performance.

Next: In Module 5, we'll explore monitoring and analytics systems, covering metrics collection, alerting, logging, and performance analysis for MEV operations.