Stock Market Metrics Computation Architecture – Complete AWS

This architecture provides a robust, scalable solution for real-time and historical stock market data processing with strategy evaluation across 500+ trading strategies using 1-minute candle data. The system ingests market data, processes it through computational phases, evaluates strategies, and delivers alerts/orders through a notification service network.

It is a layered architecture and each layer can be treated as a subnet within custom vpc.

Key Components:

1. **Data Ingestion Layer**:
    - WebSocket connections for real-time market data streaming into Kinesis Data Streams
    - Amazon S3 for historical data storage (CSV, PDFs)
    - AWS Certificate Manager for SSL/TLS security
2. **Computation Layer**:
    - AWS Fargate for containerized strategy evaluation (scalable and serverless)
    - AWS Batch for historical data processing
    - Aurora PostgreSQL (Serverless) as sync storage for strategy state
3. **Notification & Order Processing**:
    - SNS for alert distribution with topics
    - SQS queues (Order Alerts and Webhook Alerts) for reliable message delivery
    - Lambda functions for broker API integration

4. **Operational Excellence**:
    - CloudWatch for monitoring and logging
    - X-Ray for distributed tracing and debugging
    - VPC for network isolation and security
5. **Infrastructure as Code**:
    - Terraform for provisioning and lifecycle management
    - IGW for controlled internet access

IaC Terraform Implementation Strategy :

Modules:

1) Networking

- **VPC**
  Security Groups
- **Load Balancers**

2) Data Ingestion

- **Kinesis**
- **S3**

3) Compute

- **Fargate**
- **Lambda**
- **Batch**

4)Database

- **Aurora with Postgres**

5)Notification

- **SNS**
- **SQS**

6)Monitoring

- **Cloudwatch**
- **Xray**

7)Security

- **IAM**
- **Cert-manager**

Environment Strategy

1. **Workspace Approach:**
    - **dev, staging, prod workspaces for environment isolation**
    - **Shared modules with environment-specific variables**
2. **State Management:**
    - **Remote state in S3 with DynamoDB locking**
    - **Separate state files per environment**
    - **State isolation between modules where appropriate**
3. **Variable Strategy:**
    - **terraform.tfvars for common variables**
    - **dev.tfvars, staging.tfvars, prod.tfvars for environment specifics**

    - **Sensitive variables in AWS Secrets Manager with Terraform data sources**

Technology Explained:

1. **AWS Fargate over ECS/EKS**:
    - Serverless operation reduces management overhead
    - Automatic scaling matches strategy evaluation workload
    - Cost-effective for bursty computation patterns
2. **Aurora PostgreSQL Serverless**:
    - Auto-scaling matches trading session patterns
    - Cost optimization during off-market hours
    - PostgreSQL's advanced analytical capabilities
3. **Kinesis over Kafka**:
    - Fully managed service reduces operational burden
    - Seamless integration with AWS analytics services
    - Built-in scaling for variable market data volumes
4. **SNS/SQS over direct API calls**:
    - Decouples strategy evaluation from broker integration
    - Provides retry mechanisms and dead-letter handling
    - Enables multiple alert consumers without modification
5. **Terraform over CloudFormation/CDK**:
    - Multi-cloud potential for future expansion

    - Rich module ecosystem for financial services patterns
    - Better state management for complex infrastructures

Security, Scalability, and Availability

Security Implementation

1. **Data Protection**:
    - TLS everywhere (Certificate Manager)
    - VPC isolation with security groups limiting east-west traffic
    - IAM roles with least privilege (per-service roles)
    - Secrets management via AWS Secrets Manager
2. **Network Security**:
    - VPC endpoints for AWS services to avoid public internet
    - Web Application Firewall (WAF) on load balancers
    - Security groups with minimum necessary ports
3. **Operational Security**:
    - CloudTrail enabled for all API calls
    - Config Rules for compliance monitoring
    - Regular security scans of container images

Scalability Patterns

1. **Horizontal Scaling**:
    - Fargate services auto-scale based on SQS queue depth
    - Kinesis shards adjust based on incoming data rate

- Aurora Serverless scales compute based on demand

2. **Decoupled Architecture**:
    - SQS buffers between computation and notification
    - Lambda concurrency limits prevent broker API overload
    - Batch job arrays for parallel historical processing
3. **Performance Optimization**:
    - Read replicas for Aurora during heavy analysis
    - Data partitioning in S3 by date/symbol
    - Caching layer potential (ElastiCache) for frequent indicators

Availability Design

1. **Multi-AZ Deployment**:
    - Aurora across 3 AZs
    - Fargate tasks distributed across AZs
    - S3 with cross-region replication for critical data
2. **Fault Tolerance**:
    - Dead-letter queues for failed messages
    - Circuit breakers in Lambda functions
    - Retry logic with exponential backoff
3. **Disaster Recovery**:
    - Terraform modules support multi-region deployment
    - Regular S3 backups with versioning

    - Automated failover testing procedures

Modules:

Networking:

```
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = “”

  name = "trading-vpc"
  cidr = var.vpc_cidr
  azs            = var.availability_zones
  private_subnets = var.private_subnets
  public_subnets  = var.public_subnets

  enable_nat_gateway = true
  single_nat_gateway = false
}
```

Fargate Service:

```
module "strategy_evaluator" {
  source = "./modules/compute/fargate"

  cluster_name       = "strategy-evaluation"
  service_name       = "evaluator"
  task_cpu           = 4096
  task_memory        = 8192
  container_image    = var.evaluator_image
  desired_count      = var.environment == "prod" ? 3 : 1

  vpc_id             = module.vpc.vpc_id
  subnets            = module.vpc.private_subnets
  security_group_ids =
[module.sg_strategy_evaluator.security_group_id]

  sqs_queue_arns     = [module.order_alerts.queue_arn]
  db_secret_arn      =
aws_secretsmanager_secret_version.db_credentials.arn
}
```

Notification Pipeline:

```
module "trading_alerts" {
  source = "./modules/notifications"

  alert_topics = {
    "high-priority" = { protocol = "sqs" },
    "medium-priority" = { protocol = "lambda" },
    "low-priority" = { protocol = "email" }
  }

  lambda_config = {
    runtime     = "python3.12"
    handler     = "alert_processor.lambda_handler"
    source_dir  = "../src/lambda/alert_processor"
    memory_size = 512
    timeout     = 30
  }
}
```