LLM DevOps Repository Analysis Framework

Core Analysis Chain

1. Application Classification

Primary Prompt:

```
Analyze this repository and classify the application type. Respond in JSON format:
{
  "app_type": "web_app|api_service|mobile_backend|cli_tool|library|microservice|monolith",
  "primary_language": "javascript|python|java|go|rust|...",
  "framework": "react|django|spring|express|fastapi|...",
  "confidence": 0.95,
  "reasoning": "Found React components, Express server, suggests full-stack web app"
}
```

Action Logic:

```
IF confidence >= 0.8:
  AUTO_SELECT(app_type)
  TRIGGER_FOLLOW_UP_CHAINS(app_type)
ELSE:
  PRESENT_OPTIONS_TO_USER()
```

2. Database Analysis Chain

Primary Prompt:

```
Does this application require databases? Analyze the codebase and respond:
{
  "needs_database": true|false,
  "database_types": ["postgresql", "redis", "mongodb"],
  "usage_patterns": {
    "postgresql": "primary_data_storage",
    "redis": "caching_sessions",
    "mongodb": "document_storage"
  },
  "confidence": 0.85,
  "evidence": ["found SQLAlchemy imports", "Redis client configuration", "user
authentication code"]
}
```

Follow-up Chain (if needs_database = true):

```
For each detected database, ask:
"What are the specific requirements for {database_type}? Consider:
- Expected data volume (small/medium/large)
- Read/write patterns (read-heavy/write-heavy/balanced)
- Backup requirements
- High availability needs
- Performance requirements"

Response format:
{
  "database": "postgresql",
  "volume": "medium",
  "pattern": "read_heavy",
  "ha_required": true,
  "backup_strategy": "daily_automated",
  "confidence": 0.75
}
```

3. Storage Requirements Chain

Primary Prompt:

```
Analyze file storage and static asset requirements:
{
  "needs_object_storage": true|false,
  "storage_types": ["s3_bucket", "cdn", "local_files"],
  "use_cases": {
    "s3_bucket": ["user_uploads", "backup_storage"],
    "cdn": ["static_assets", "media_files"],
    "local_files": ["temporary_processing"]
  },
  "estimated_volume": "small|medium|large",
  "public_access": true|false,
  "confidence": 0.80
}
```

Follow-up Chain (if needs_object_storage = true):

```
"What are the specific S3/storage configurations needed?
- Bucket permissions (public/private)
- CORS requirements
- Lifecycle policies
- Access patterns
- Security requirements"
```

4. Networking & Security Chain

Primary Prompt:

```
Analyze networking and security requirements:
{
  "internet_facing": true|false,
  "load_balancer_needed": true|false,
  "ssl_required": true|false,
  "cors_requirements": true|false,
  "authentication_method": "oauth|jwt|session|none",
  "external_apis": ["stripe", "sendgrid", "aws"],
  "internal_services": 2,
  "confidence": 0.90
}
```

Branching Logic:

```
IF internet_facing = true:
  TRIGGER: Security_Hardening_Chain()
  TRIGGER: SSL_Certificate_Chain()
  TRIGGER: WAF_Requirements_Chain()

IF load_balancer_needed = true:
  TRIGGER: Load_Balancer_Config_Chain()
```

5. Microservices Architecture Chain

Primary Prompt:

```
Is this a microservices architecture? Analyze:
{
  "is_microservices": true|false,
  "service_count": 3,
  "services": [
    {
      "name": "user-service",
      "type": "api",
      "dependencies": ["database", "redis"]
    },
    {
      "name": "notification-service",
      "type": "worker",
      "dependencies": ["message_queue"]
    }
  ],
  "service_communication": "rest|grpc|message_queue",
  "service_discovery": "kubernetes|consul|none",
  "confidence": 0.85
}
```

Follow-up Chain (if is_microservices = true):

```
For each service, analyze:
"What are the deployment requirements for {service_name}?
- Resource requirements (CPU/Memory)
- Scaling needs
- Health check endpoints
- Dependencies
- Environment variables"
```

6. Message Queue & Event Streaming Chain

Primary Prompt:

```
Does this application require message queues or event streaming?
{
  "needs_messaging": true|false,
  "messaging_types": ["rabbitmq", "kafka", "sqs", "redis_pub_sub"],
  "patterns": {
    "rabbitmq": "task_queue",
    "kafka": "event_streaming",
    "sqs": "async_processing"
  },
  "message_volume": "low|medium|high",
  "durability_required": true|false,
  "confidence": 0.75
}
```

7. Caching Strategy Chain

Primary Prompt:

```
What caching strategies are needed?
{
  "needs_caching": true|false,
  "cache_types": ["redis", "memcached", "application_cache"],
  "cache_patterns": {
    "redis": ["session_store", "api_cache"],
    "application_cache": ["computed_results"]
  },
  "cache_size": "small|medium|large",
  "ttl_requirements": "short|medium|long",
  "confidence": 0.80
}
```

8. Observability & Monitoring Chain

Primary Prompt:

```
What monitoring and observability is implemented or needed?
{
  "current_monitoring": ["logs", "basic_metrics"],
  "missing_monitoring": ["tracing", "alerting", "dashboards"],
  "log_volume": "low|medium|high",
  "metrics_needed": ["business", "technical", "security"],
  "alerting_requirements": {
    "error_rate": "threshold_5_percent",
    "response_time": "threshold_500ms",
    "availability": "threshold_99_percent"
  },
  "compliance_requirements": ["gdpr", "hipaa", "sox"],
  "confidence": 0.70
}
```

9. CI/CD Pipeline Chain

Primary Prompt:

```
Analyze CI/CD requirements based on the codebase:
{
  "has_existing_ci": true|false,
  "existing_tools": ["github_actions", "docker"],
  "build_requirements": {
    "node_version": "18",
    "python_version": "3.9",
    "build_time": "medium"
  },
  "test_strategy": {
    "unit_tests": true,
    "integration_tests": false,
    "e2e_tests": true
  },
  "deployment_strategy": "blue_green|rolling|canary",
  "environments": ["dev", "staging", "prod"],
  "confidence": 0.85
}
```

10. Security Requirements Chain

Primary Prompt:

```
What security measures are implemented or needed?
{
  "current_security": ["input_validation", "authentication"],
  "missing_security": ["rate_limiting", "security_headers", "secrets_management"],
  "sensitive_data": ["user_passwords", "api_keys", "pii"],
  "compliance_needs": ["gdpr", "ccpa"],
  "vulnerability_scan": "needed",
  "secrets_management": "env_vars|vault|k8s_secrets",
  "confidence": 0.75
}
```

Chain-of-Thought Framework Implementation

Response Handler Pattern:

javascript

```javascript
class DevOpsAnalysisChain {
  async analyzeRepository(repoContents) {
    const results = {};

    // 1. Primary Classification
    const appType = await this.classifyApplication(repoContents);
    results.appType = appType;

    // 2. Conditional Branching
    if (appType.confidence >= 0.8) {
      // Trigger relevant chains based on app type
      const chains = this.getRelevantChains(appType.app_type);

      for (const chain of chains) {
        const result = await this.executeChain(chain, repoContents, results);
        results[chain.name] = result;

        // Conditional follow-ups
        if (result.needs_follow_up) {
          const followUp = await this.executeFollowUpChain(
            chain.follow_up_prompt,
            result,
            repoContents
          );
          results[`${chain.name}_details`] = followUp;
        }
      }
    }

    return this.generateTerraformRecommendations(results);
  }

  getRelevantChains(appType) {
    const chainMap = {
      'web_app': ['database', 'storage', 'networking', 'caching', 'monitoring'],
      'api_service': ['database', 'networking', 'caching', 'monitoring', 'security'],
      'microservice': ['database', 'networking', 'messaging', 'monitoring', 'service_mesh'],
      'cli_tool': ['storage', 'security'],
      'mobile_backend': ['database', 'storage', 'networking', 'push_notifications']
    };

    return chainMap[appType] || ['database', 'storage', 'networking'];
  }
}
```

Confidence-Based Action Logic:

javascript

```javascript
function processAnalysisResult(result) {
  if (result.confidence >= 0.9) {
    return { action: 'AUTO_APPLY', value: result };
  } else if (result.confidence >= 0.7) {
    return { action: 'SUGGEST_WITH_OPTION', value: result };
  } else if (result.confidence >= 0.5) {
    return { action: 'PRESENT_OPTIONS', value: result };
  } else {
    return { action: 'REQUIRE_USER_INPUT', value: null };
  }
}
```

Branching Decision Tree:

javascript

```javascript
const decisionTree = {
  database: {
    condition: (result) => result.needs_database === true,
    next_chains: ['database_sizing', 'backup_strategy', 'ha_requirements']
  },
  microservices: {
    condition: (result) => result.is_microservices === true,
    next_chains: ['service_mesh', 'service_discovery', 'inter_service_auth']
  },
  high_traffic: {
    condition: (result) => result.expected_traffic === 'high',
    next_chains: ['load_balancing', 'auto_scaling', 'caching']
  }
};
```

Final Terraform Generation Chain

Final Prompt:

```
Based on all analysis results, generate Terraform resource recommendations:
{
  "terraform_modules": [
    {
      "module": "aws_rds_postgresql",
      "configuration": {
        "instance_class": "db.t3.medium",
        "storage": "100GB",
        "backup_retention": "7_days"
      },
      "reasoning": "Medium traffic web app with user data"
    }
  ],
  "estimated_cost": "$200_monthly",
  "complexity": "medium",
  "deployment_time": "2_hours"
}
```

This framework provides:

1. **Systematic Analysis** - Each aspect analyzed in order
2. **Conditional Branching** - Follow-up questions based on results
3. **Confidence Scoring** - Automated vs manual decisions
4. **Chain Dependencies** - Results inform subsequent analysis
5. **Final Synthesis** - All results combined for Terraform generation