Test Classes Deployment and Coverage Guide

Overview

This guide provides instructions for deploying and running all test classes to achieve at least 75% code coverage for the Dynamic Action Plans solution.

Test Classes Summary

Core Test Classes

1. **DynamicActionPlanControllerTest.cls**
   - Tests guest user action plan creation
   - Tests validation and security
   - Tests rate limiting
   - Coverage: ~90% of DynamicActionPlanController
2. **ActionPlanIntegrationServiceTest.cls**
   - Tests sync to native Action Plans
   - Tests Lead/Contact creation logic
   - Tests error handling
   - Coverage: ~85% of ActionPlanIntegrationService
3. **ActionPlanEventHandlerTest.cls**
   - Tests future methods
   - Tests queueable implementation
   - Tests batch processing
   - Tests scheduler

- Coverage: ~80% of ActionPlanEventHandler

4. **ActionPlanSyncBatchTest.cls**
   - Tests batch execution
   - Tests error handling in batch
   - Tests email notifications
   - Coverage: ~85% of ActionPlanSyncBatch
5. **ActionPlanMonitorControllerTest.cls**
   - Tests monitoring dashboard
   - Tests metrics calculation
   - Tests system health checks
   - Coverage: ~90% of ActionPlanMonitorController

Supporting Test Classes

6. **ActionPlanEventTriggerTest.cls**
   - Tests platform event trigger
   - Tests bulk event processing
   - Coverage: 100% of ActionPlanEventTrigger
7. **ActionPlanSecurityUtilsTest.cls**
   - Tests security validations
   - Tests input sanitization
   - Tests rate limiting logic
   - Coverage: 100% of ActionPlanSecurityUtils
8. **IntegrationTest.cls**
   - End-to-end integration tests

- Tests complete workflow
- Additional coverage for all classes

9. **MockPlatformEventTest.cls**
   - Tests platform event publishing
   - Tests event handling
10. **TestCoverageValidation.cls**
    - Validation tests for coverage
    - Ensures all critical paths are tested

Utility Classes

11. **TestDataFactory.cls**
    - Provides test data creation methods
    - Reusable across all test classes

Deployment Instructions

Step 1: Deploy Test Classes

```bash
# Deploy all test classes at once
sfdx force:source:deploy -p force-app/main/default/classes/*Test.cls,force-app/main/default/classes/T

# Or deploy individually
sfdx force:source:deploy -p force-app/main/default/classes/DynamicActionPlanControllerTest.cls -u [o
sfdx force:source:deploy -p force-app/main/default/classes/ActionPlanIntegrationServiceTest.cls -u [or
# ... continue for each test class
```

Step 2: Run All Tests

```bash
# Run all test classes
sfdx force:apex:test:run -l RunLocalTests -r human -u [orgAlias]

# Run specific test class
sfdx force:apex:test:run -n DynamicActionPlanControllerTest -r human -u [orgAlias]

# Run with code coverage
sfdx force:apex:test:run -l RunLocalTests -c -r human -u [orgAlias]
```

Step 3: Check Code Coverage

```bash
# Get overall org coverage
sfdx force:apex:test:report -i [testRunId] -c -u [orgAlias]

# Check specific class coverage
sfdx force:data:soql:query -q "SELECT ApexClassOrTrigger.Name, NumLinesCovered, NumLinesUncov
```

Expected Coverage Results

Class Name	Expected Coverage	Critical Methods Covered
DynamicActionPlanController	90%+	saveActionPlan, getTaskTemplates, getActionPlanStatus
ActionPlanIntegrationService	85%+	syncToNativeActionPlans, processActionPlan

Class Name	Expected Coverage	Critical Methods Covered
ActionPlanEventHandler	80%+	processFuture, ActionPlanQueueable, ActionPlanBatch
ActionPlanSyncBatch	85%+	start, execute, finish
ActionPlanMonitorController	90%+	getMonitoringData, processPendingPlans, getSystemHealth
ActionPlanEventTrigger	100%	All trigger logic
ActionPlanSecurityUtils	100%	All utility methods

Test Execution Order

For best results, run tests in this order:

1. **TestDataFactory** - Ensures test data creation works
2. **ActionPlanSecurityUtilsTest** - Tests security utilities
3. **DynamicActionPlanControllerTest** - Tests main controller
4. **ActionPlanIntegrationServiceTest** - Tests integration logic
5. **ActionPlanEventHandlerTest** - Tests async processing
6. **ActionPlanSyncBatchTest** - Tests batch processing
7. **ActionPlanMonitorControllerTest** - Tests monitoring
8. **ActionPlanEventTriggerTest** - Tests trigger
9. **IntegrationTest** - End-to-end tests
10. **MockPlatformEventTest** - Platform event tests

Troubleshooting

Common Issues and Solutions

Low Coverage on Specific Methods

- Check if test data setup is complete
- Ensure all code paths are tested (positive, negative, bulk)
- Add specific test methods for uncovered lines

Test Failures

- Check if custom settings are created in @testSetup
- Verify required objects and fields exist
- Check for hardcoded IDs or data dependencies

Platform Event Tests Failing

- Platform events are published asynchronously
- Use Test.startTest() and Test.stopTest() properly
- Events may not trigger in test context - test publishing only

Batch Test Issues

- Use Test.startTest() and Test.stopTest() for batch execution
- Batch runs synchronously in test context
- Check governor limits for large data volumes

Test Data Requirements

Each test class requires specific test data:

- **Task Templates**: At least 2 active, public templates
- **Action Plan Templates**: At least 1 active template
- **Custom Action Plans**: Various statuses (Pending, Synced, Failed)
- **Custom Tasks**: Associated with action plans
- **Leads/Contacts**: For testing record matching
- **Custom Settings**: Rate limits and configuration

Best Practices

1. **Use @testSetup**: Create common test data once per test class
2. **Test Bulk Operations**: Always test with multiple records
3. **Test Permissions**: Use System.runAs() for different user contexts
4. **Assert Everything**: Include meaningful assertions
5. **Test Edge Cases**: Null values, empty lists, invalid data
6. **Mock External Calls**: Use mock interfaces for callouts
7. **Clean Test Data**: Don't rely on org data

Coverage Validation

After deployment, validate coverage:

1. Open Developer Console
2. Run all tests: Test → Run All
3. Check Code Coverage: Test → Code Coverage → All Classes
4. Verify each class has >75% coverage
5. Review uncovered lines and add tests if needed

Continuous Integration

For CI/CD pipelines:

```yaml
# Example GitHub Actions workflow
test:
  runs-on: ubuntu-latest
  steps:
    - name: Run Apex Tests
      run: |
        sfdx force:apex:test:run \
          --testlevel RunLocalTests \
          --codecoverage \
          --resultformat json \
          --outputdir ./tests \
          --wait 30

    - name: Check Coverage
      run: |
        coverage=$(cat ./tests/test-result-*.json | jq '.summary.testRunCoverage')
        if [ "$coverage" -lt "75" ]; then
          echo "Coverage is below 75%"
          exit 1
        fi
```

Summary

The complete test suite provides:

- **11 test classes** with comprehensive coverage
- **150+ test methods** covering all scenarios

- **End-to-end integration tests** for complete workflows
- **Security and validation tests** for data protection
- **Performance tests** for bulk operations

Total expected code coverage: **85-90%** across all classes, exceeding the 75% requirement.