Development and Testing Guide

Complete Development Workflow for Service Function Architecture

---

Table of Contents

Executive Summary

This guide covers development and testing for the Service Function Architecture platform using a single-node RKE2 cluster that provides a complete, production-identical environment on developer workstations.

Key Features:

- **Single-Node RKE2 Cluster**: Complete platform on developer machines
- **Rapid Restart Development**: 3-5 second deployments with full container isolation
- **Real Platform Testing**: No mocking required
- **Cross-Language Support**: Schema-first development with automatic code generation

Benefits:

- Complete environment setup in minutes
- Fast feedback loops (3-5 second deployments)
- Production-identical behavior
- Zero-configuration debugging

Quick Start

Installation

```bash
# Install complete development environment
curl -sSL https://install.platform.dev/dev | bash

# Verify installation
platform-cli cluster status
```

Create Your First Service

```bash
# Create new service
platform-cli create service UserService --language csharp

# Start development
cd UserService
platform-cli dev start --watch
```

Development Environment Setup

Hardware Requirements

Minimum Configuration:

- CPU: 8 cores
- Memory: 16GB RAM
- Storage: 100GB SSD
- OS: Windows 10/11, macOS 11+, Ubuntu 20.04+

Recommended Configuration:

- CPU: 12+ cores
- Memory: 32GB RAM
- Storage: 250GB NVMe

Platform Services Configuration

yaml

```yaml
# Development resource allocation
resources:
 emqx:
  memory: "1-2Gi"   # vs 4-8Gi in production
  cpu: "0.5-1"      # vs 2-4 in production
  replicas: 1       # vs 3+ in production

 postgresql:
  memory: "2-4Gi"   # vs 8-16Gi in production
  cpu: "1-2"        # vs 4-8 in production
  replicas: 1       # vs 3 in production

 mongodb:
  memory: "2-4Gi"   # vs 8-16Gi in production
  cpu: "1-2"        # vs 4-8 in production
  replicas: 1       # vs 3+ in production
```

Development Cluster Configuration

Platform Configuration File (`platform.dev.yaml`):

yaml

```yaml
platform:
 environment: development
 cluster:
  type: rke2
  nodes: 1
  resources:
   cpu: "8-16"
   memory: "16-32Gi"
   storage: "100Gi"

 services:
  emqx:
   enabled: true
   replicas: 1
   resources:
    memory: "2Gi"
    cpu: "1"

  postgresql:
   enabled: true
   replicas: 1
   resources:
    memory: "4Gi"
    cpu: "2"
   persistence: true

  mongodb:
   enabled: true
   replicas: 1
   resources:
    memory: "4Gi"
    cpu: "2"
   persistence: true

  redis:
   enabled: true
   replicas: 1
   resources:
    memory: "1Gi"
    cpu: "0.5"

 development:
  features:
```

```
    rapidRestart: true
    hotReload: true
    debugDashboard: true
    schemaValidation: true
    liveReload: true
    performanceMonitoring: true

  tools:
    enabled:
      - codeGeneration
      - schemaEditor

  testing:
    enabled: true
    integrationTests: true
    mockServices: false # Use real services
    testDataGeneration: true

schema-registry:
  enabled: true
  validation: strict
  codeGeneration:
    enabled: true
    languages: ["csharp", "javascript", "python", "go", "java"]
    outputPath: "./generated"

local-registry:
  enabled: true
  port: 5000
  persistence: false # Use memory for speed

operations:
  monitoring:
    interval: 10s     # More frequent in dev
    retention: 7d     # Shorter retention
    metrics:
      - cpu
      - memory
      - messageQueues
      - responseTime
      - errorRate
```

Development Cluster Validation

bash

```
# Check cluster status
platform-cli cluster status

# Validate all services
platform-cli services health

# Output:
# ✅ Kubernetes cluster: Ready (1 node)
# ✅ EMQX broker: Ready (1 instance)
# ✅ PostgreSQL: Ready (1 instance)
# ✅ MongoDB: Ready (1 instance)
# ✅ Redis: Ready (1 instance)
# ✅ Rapid Restart: Active
# ✅ Debug Dashboard: http://localhost:8080
```

Development Workflow

Project Structure and Templates

Service Function Project Structure:

```
my-service/
├── src/
│   ├── Services/
│   │   ├── UserService.cs
│   │   ├── OrderService.cs
│   │   └── PaymentService.cs
│   ├── Models/
│   │   ├── UserModels.cs
│   │   └── OrderModels.cs
│   └── Program.cs
├── schemas/
│   ├── UserService.openrpc.json
│   ├── OrderService.openrpc.json
│   └── PaymentService.openrpc.json
├── tests/
│   ├── UserServiceTests.cs
│   ├── OrderServiceTests.cs
│   └── IntegrationTests.cs
├── platform.yaml
├── Dockerfile
└── README.md
```

Schema-First Development

1. Define Schema (`schemas/UserService.openrpc.json`):

json

```
{
 "openrpc": "1.3.0",
 "info": { "title": "UserService", "version": "1.0.0" },
 "methods": [
  {
   "name": "createUser",
   "params": [
    {
     "name": "userData",
     "schema": { "$ref": "#/components/schemas/CreateUserRequest" }
    }
   ],
   "result": {
    "schema": { "$ref": "#/components/schemas/CreateUserResponse" }
   }
  }
 ],
 "components": {
  "schemas": {
   "CreateUserRequest": {
    "type": "object",
    "properties": {
     "email": { "type": "string", "format": "email" },
     "name": { "type": "string" }
    },
    "required": ["email", "name"]
   },
   "CreateUserResponse": {
    "type": "object",
    "properties": {
     "userId": { "type": "string" },
     "success": { "type": "boolean" }
    }
   }
  }
 }
}
```

2. Generate Code:

bash

```
platform-cli generate service --schema schemas/UserService.openrpc.json
```

3. Implement Business Logic (`src/UserService.cs`):

csharp

```
public partial class UserService
{
    public async Task<CreateUserResponse> CreateUser(CreateUserRequest userData)
    {
        var userId = await _dataStore.Insert("users", new JsonObject
        {
            ["email"] = userData.Email,
            ["name"] = userData.Name,
            ["createdAt"] = DateTime.UtcNow
        });

        return new CreateUserResponse { UserId = userId, Success = true };
    }
}
```

Rapid Restart Development

Development Loop:

1. Edit code→ Auto-detection→ Build → Deploy → Test (3-5 seconds)

bash

```
# Start with file watching
platform-cli dev start --watch

# Output after code change:
# 📝 File changed: src/UserService.cs
# 🔨 Building...
# 🚀 Deploying...
# ✅ Restart complete (3.2s)
```

Complete Cross-Language Example

User A: Python Service Development

Python Service (`user_service.py`):

python

```python
from platform_core import ServiceFunction, Function
from typing import Optional
from dataclasses import dataclass
from datetime import datetime

@dataclass
class CreateUserRequest:
  """Request to create a new user"""
  email: str # User's email address - must be unique
  name: str  # User's full name
  department: Optional[str] = None # Optional department assignment

@dataclass
class CreateUserResponse:
  """Response from user creation"""
  user_id: str  # Unique identifier for the created user
  success: bool  # Indicates whether the operation succeeded
  message: Optional[str] = None # Optional success/error message

@ServiceFunction("UserManagement")
class UserService:
  def __init__(self, data_store, messaging, authorizer):
    self.data_store = data_store
    self.messaging = messaging
    self.authorizer = authorizer

  @Function("createUser")
  async def create_user(self, request: CreateUserRequest) -> CreateUserResponse:
    """Create a new user account"""
    try:
      # Check for existing user
      existing_users = await self.data_store.query(
        "users",
        {"email": request.email}
      )

      if existing_users:
        return CreateUserResponse(
          user_id="",
          success=False,
          message="User with this email already exists"
        )
```

```
        # Create new user
        user_data = {
            "email": request.email,
            "name": request.name,
            "department": request.department,
            "created_at": datetime.utcnow().isoformat(),
            "active": True
        }

        user_id = await self.data_store.insert("users", user_data)

        return CreateUserResponse(
            user_id=user_id,
            success=True,
            message="User created successfully"
        )

    except Exception as e:
        return CreateUserResponse(
            user_id="",
            success=False,
            message=f"Error creating user: {str(e)}"
        )
```

Auto-Generated OpenRPC Schema

The platform automatically generates this schema from Python type annotations:

json

```
{
 "openrpc": "1.3.0",
 "info": {
  "title": "UserManagement",
  "version": "1.0.0"
 },
 "methods": [
  {
   "name": "createUser",
   "summary": "Create a new user account",
   "params": [
    {
     "name": "request",
     "description": "Request to create a new user",
     "required": true,
     "schema": { "$ref": "#/components/schemas/CreateUserRequest" }
    }
   ],
   "result": {
    "schema": { "$ref": "#/components/schemas/CreateUserResponse" }
   }
  }
 ],
 "components": {
  "schemas": {
   "CreateUserRequest": {
    "type": "object",
    "properties": {
     "email": {
      "type": "string",
      "format": "email",
      "description": "User's email address - must be unique"
     },
     "name": {
      "type": "string",
      "description": "User's full name"
     },
     "department": {
      "type": "string",
      "description": "Optional department assignment"
     }
    },
    "required": ["email", "name"]
```

```
    },
    "CreateUserResponse": {
      "type": "object",
      "properties": {
        "user_id": {
          "type": "string",
          "description": "Unique identifier for the created user"
        },
        "success": {
          "type": "boolean",
          "description": "Indicates whether the operation succeeded"
        },
        "message": {
          "type": "string",
          "description": "Optional success/error message"
        }
      },
      "required": ["user_id", "success"]
    }
  }
 }
}
```

User B: C# Client Generation

Generate C# Client:

```
bash

platform-cli generate client --service UserManagement --language csharp --output ./Generated
```

Generated C# Models:

csharp

```csharp
// Generated/UserManagement/Models.cs
namespace Generated.UserManagement
{
    /// <summary>
    /// Request to create a new user
    /// </summary>
    public class CreateUserRequest
    {
        /// <summary>
        /// User's email address - must be unique
        /// </summary>
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        /// <summary>
        /// User's full name
        /// </summary>
        [Required]
        public string Name { get; set; }

        /// <summary>
        /// Optional department assignment
        /// </summary>
        public string? Department { get; set; }
    }

    /// <summary>
    /// Response from user creation
    /// </summary>
    public class CreateUserResponse
    {
        public string UserId { get; set; }
        public bool Success { get; set; }
        public string? Message { get; set; }
    }
}
```

Generated C# Client:

csharp

```csharp
// Generated/UserManagement/UserManagementClient.cs
public class UserManagementClient : IUserManagementClient
{
    private readonly IMessaging _messaging;
    private const string ServiceTopic = "service/UserManagement";

    public UserManagementClient(IMessaging messaging)
    {
        _messaging = messaging;
    }

    public async Task<CreateUserResponse> CreateUserAsync(CreateUserRequest request)
    {
        var message = new Message
        {
            Body = new
            {
                function = "createUser",
                parameters = request
            }
        };

        var response = await _messaging.SendAndWait(
            ServiceTopic,
            message,
            TimeSpan.FromSeconds(30));

        return response.Body.ToObject<CreateUserResponse>();
    }
}
```

User B's Application:

```csharp
// Program.cs
class Program
{
    static async Task Main(string[] args)
    {
        var userClient = serviceProvider.GetRequiredService<IUserManagementClient>();

        var response = await userClient.CreateUserAsync(new CreateUserRequest
        {
            Email = "john.doe@example.com",
            Name = "John Doe",
            Department = "Engineering"
        });

        if (response.Success)
        {
            Console.WriteLine($"User created: {response.UserId}");
        }
        else
        {
            Console.WriteLine($"Error: {response.Message}");
        }
    }
}
```

Comprehensive Testing Framework

Test Platform Configuration

Test Base Class:

csharp

```csharp
// Platform.Testing framework
public abstract class ServiceTestBase<T> : IClassFixture<TestPlatformFixture>
    where T : class
{
    protected readonly TestPlatformFixture Fixture;
    protected readonly T Service;
    protected readonly IDataStore DataStore;
    protected readonly IMessaging Messaging;

    protected ServiceTestBase(TestPlatformFixture fixture)
    {
        Fixture = fixture;
        Service = fixture.ServiceProvider.GetRequiredService<T>();
        DataStore = fixture.ServiceProvider.GetRequiredService<IDataStore>();
        Messaging = fixture.ServiceProvider.GetRequiredService<IMessaging>();
    }

    protected async Task<UserContext> CreateTestUser(string email = "test@example.com")
    {
        return new UserContext
        {
            UserId = Guid.NewGuid().ToString(),
            Email = email,
            Roles = new List<string> { "user" }
        };
    }
}
```

Test Platform Fixture:

```csharp
public class TestPlatformFixture : IDisposable
{
    public IServiceProvider ServiceProvider { get; private set; }

    public TestPlatformFixture()
    {
        var services = new ServiceCollection();

        // Configure test services
        services.AddSingleton<IDataStore>(provider =>
            new TestDataStore(connectionString: "test-database"));

        services.AddSingleton<IMessaging>(provider =>
            new TestMessaging(brokerUrl: "mqtt://localhost:1883"));

        services.AddTransient<UserService>();

        ServiceProvider = services.BuildServiceProvider();
        InitializeTestEnvironment().Wait();
    }

    private async Task InitializeTestEnvironment()
    {
        var dataStore = ServiceProvider.GetRequiredService<IDataStore>();
        await dataStore.CreateCollection("users");
    }

    public void Dispose()
    {
        CleanupTestEnvironment().Wait();
        ServiceProvider?.Dispose();
    }

    private async Task CleanupTestEnvironment()
    {
        var dataStore = ServiceProvider.GetRequiredService<IDataStore>();
        await dataStore.Delete("users", Query.All());
    }
}
```

Unit Tests

csharp

```csharp
public class UserServiceTests : ServiceTestBase<UserService>
{
    [Fact]
    public async Task CreateUser_ValidData_ReturnsSuccess()
    {
        var request = new CreateUserRequest
        {
            Email = "test@example.com",
            Name = "Test User"
        };

        var result = await Service.CreateUser(request);

        Assert.True(result.Success);
        Assert.NotNull(result.UserId);

        // Verify data was stored
        var storedUser = await DataStore.GetById("users", result.UserId);
        Assert.Equal(request.Email, storedUser["email"].ToString());
    }

    [Fact]
    public async Task CreateUser_DuplicateEmail_ReturnsError()
    {
        var userData = new CreateUserRequest
        {
            Email = "duplicate@example.com",
            Name = "First User"
        };

        await Service.CreateUser(userData); // Create first user

        var duplicateData = new CreateUserRequest
        {
            Email = "duplicate@example.com",
            Name = "Second User"
        };

        await Assert.ThrowsAsync<BusinessLogicException>(
            () => Service.CreateUser(duplicateData)
        );
    }
}
```

Integration Tests

csharp

```csharp
public class UserManagementIntegrationTests : IClassFixture<TestPlatformFixture>
{
    private readonly TestPlatformFixture _fixture;

    public UserManagementIntegrationTests(TestPlatformFixture fixture)
    {
        _fixture = fixture;
    }

    [Fact]
    public async Task UserCreation_EndToEnd_WorksCorrectly()
    {
        var messaging = _fixture.ServiceProvider.GetRequiredService<IMessaging>();

        var request = new Message
        {
            Body = new {
                function = "createUser",
                parameters = new {
                    email = "integration@example.com",
                    name = "Integration Test User"
                }
            }
        };

        var response = await messaging.SendAndWait(
            "service/UserManagement",
            request,
            TimeSpan.FromSeconds(10));

        Assert.Equal("success", response.Body["status"].ToString());

        // Verify data persistence
        var userId = response.Body["userId"].ToString();
        var dataStore = _fixture.ServiceProvider.GetRequiredService<IDataStore>();
        var user = await dataStore.GetById("users", userId);

        Assert.Equal("integration@example.com", user["email"].ToString());
    }
}
```

Performance Tests

Memory and Resource Testing:

csharp

```
public class ResourceUsageTests
{
  [Fact]
  public async Task UserService_UnderLoad_DoesNotLeakMemory()
  {
    var initialMemory = GC.GetTotalMemory(forceFullCollection: true);

    // Create 1000 users to stress test memory usage
    var tasks = new List<Task>();
    for (int i = 0; i < 1000; i++)
    {
      tasks.Add(CreateUserAsync($"memory-test-{i}@example.com"));
    }

    await Task.WhenAll(tasks);

    // Force garbage collection
    GC.Collect();
    GC.WaitForPendingFinalizers();
    GC.Collect();

    var finalMemory = GC.GetTotalMemory(forceFullCollection: true);
    var memoryIncrease = finalMemory - initialMemory;

    // Memory increase should be reasonable (< 50MB for 1000 users)
    Assert.True(memoryIncrease < 50 * 1024 * 1024,
      $"Memory usage increased by {memoryIncrease / 1024 / 1024}MB");
  }

  private async Task CreateUserAsync(string email)
  {
    var messaging = TestPlatform.GetMessaging();
    await messaging.SendAndWait(
      "service/UserManagement",
      new Message { Body = new { function = "createUser", parameters = new { email, name = "Test User"
      TimeSpan.FromSeconds(5)
    );
  }
}
```

Benchmark Testing:

```csharp
[MemoryDiagnoser]
public class UserServiceBenchmarks
{
    private UserService _userService;

    [GlobalSetup]
    public void Setup()
    {
        var fixture = new TestPlatformFixture();
        _userService = fixture.ServiceProvider.GetRequiredService<UserService>();
    }

    [Benchmark]
    public async Task CreateUser_DirectCall()
    {
        var userData = new CreateUserRequest
        {
            Email = $"benchmark-{Guid.NewGuid()}@example.com",
            Name = "Benchmark User"
        };

        await _userService.CreateUser(userData);
    }

    [Benchmark]
    [Arguments(1)]
    [Arguments(10)]
    [Arguments(100)]
    public async Task GetUser_Batch(int batchSize)
    {
        var tasks = new List<Task>();

        for (int i = 0; i < batchSize; i++)
        {
            tasks.Add(_userService.GetUser("existing-user-id"));
        }

        await Task.WhenAll(tasks);
    }
}
```

Advanced Debugging with Distributed Tracing

Tracing Setup

Service Implementation with Tracing:

csharp

```csharp
public partial class UserService
{
    private static readonly ActivitySource ActivitySource = new("UserService");

    public async Task<CreateUserResponse> CreateUser(CreateUserRequest userData)
    {
        using var activity = ActivitySource.StartActivity("CreateUser");
        activity?.SetTag("user.email", userData.Email);
        activity?.SetTag("user.department", userData.Department);

        try
        {
            // Validate business rules
            using var validationActivity = ActivitySource.StartActivity("ValidateUser");

            var existingUser = await _dataStore.Query("users",
                Query.Where("email", userData.Email)).FirstOrDefaultAsync();

            if (existingUser != null)
            {
                activity?.SetStatus(ActivityStatusCode.Error, "Duplicate email");
                throw new BusinessLogicException("User with this email already exists");
            }

            validationActivity?.SetStatus(ActivityStatusCode.Ok);

            // Create user record
            using var createActivity = ActivitySource.StartActivity("CreateUserRecord");

            var userId = await _dataStore.Insert("users", new JsonObject
            {
                ["email"] = userData.Email,
                ["name"] = userData.Name,
                ["createdAt"] = DateTime.UtcNow
            });

            createActivity?.SetTag("user.id", userId);
            createActivity?.SetStatus(ActivityStatusCode.Ok);

            activity?.SetStatus(ActivityStatusCode.Ok);

            return new CreateUserResponse { UserId = userId, Success = true };
        }
```

```
        catch (Exception ex)
        {
            activity?.SetStatus(ActivityStatusCode.Error, ex.Message);
            throw;
        }
    }
}
```

Trace Commands

```bash
# View trace data in dashboard
platform-cli debug traces --service UserManagement --operation CreateUser

# Real-time trace following
platform-cli debug traces follow --filter "user.email=test@example.com"

# Export traces for external analysis
platform-cli debug traces export --format jaeger --output traces.json
```

Multi-Service Development

Working with Multiple Services

```bash
# Start multiple services in development mode
platform-cli dev start --services UserManagement,OrderManagement,PaymentService --watch

# Test cross-service communication
platform-cli test integration --services UserManagement,OrderManagement

# Monitor cross-service message flow
platform-cli debug trace --services UserManagement,OrderManagement --follow
```

Cross-Service Integration Testing

csharp

```csharp
[Fact]
public async Task OrderCreation_WithUserService_WorksEndToEnd()
{
    // Create user first
    var userResponse = await _userClient.CreateUserAsync(new CreateUserRequest
    {
        Email = "customer@example.com",
        Name = "Test Customer"
    });

    // Create order for that user
    var orderResponse = await _orderClient.CreateOrderAsync(new CreateOrderRequest
    {
        UserId = userResponse.UserId,
        ProductId = "product-123",
        Quantity = 2
    });

    Assert.True(orderResponse.Success);
    Assert.Equal(userResponse.UserId, orderResponse.Order.UserId);
}
```

Schema Evolution and Backward Compatibility

Testing Schema Changes

csharp

```csharp
public class SchemaEvolutionTests
{
    [Fact]
    public async Task NewOptionalField_MaintainsBackwardCompatibility()
    {
        // Test with old client format against new service
        var oldRequest = new { email = "test@example.com", name = "Test User" };

        var response = await messaging.SendAndWait(
            "service/UserManagement",
            new Message { Body = new { function = "createUser", parameters = oldRequest } },
            TimeSpan.FromSeconds(10));

        Assert.True(response.Body["success"].ToObject<bool>());
    }

    [Fact]
    public async Task SchemaCompatibility_ChecksVersions()
    {
        var compatibility = await CheckSchemaCompatibility("UserManagement", "1.0.0", "1.1.0");

        Assert.True(compatibility.IsBackwardCompatible);
        Assert.Empty(compatibility.BreakingChanges);
    }
}
```

Schema Evolution Example

Version 1.0.0→ 1.1.0 (Backward Compatible):

```python
# Original
@dataclass
class CreateUserRequest:
    email: str
    name: str

# New version with optional field
@dataclass
class CreateUserRequest:
    email: str
    name: str
    phone_number: Optional[str] = None # NEW OPTIONAL FIELD
```

The platform automatically detects this as backward compatible and generates appropriate client code.

CI/CD Integration

GitHub Actions

yaml

```yaml
name: Deploy Service
on: [push]

jobs:
 test:
  runs-on: ubuntu-latest

  steps:
  - uses: actions/checkout@v4

  - name: Install Platform CLI
   run: curl -sSL https://install.platform.dev/cli | bash

  - name: Start Test Environment
   run: platform-cli dev start --minimal --wait-ready

  - name: Setup .NET
   uses: actions/setup-dotnet@v3
   with:
    dotnet-version: '8.0'

  - name: Restore dependencies
   run: dotnet restore

  - name: Build
   run: dotnet build --no-restore

  - name: Run Unit Tests
   run: dotnet test --no-build --verbosity normal --filter "Category!=Integration"

  - name: Run Integration Tests
   run: dotnet test --no-build --verbosity normal --filter "Category=Integration"

  - name: Validate Schemas
   run: platform-cli validate schemas --strict

  - name: Package Service
   run: platform-cli package --output package.tar.gz

  - name: Upload Artifacts
   uses: actions/upload-artifact@v3
   with:
    name: service-package
```

```
    path: package.tar.gz

deploy-staging:
  needs: test
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/develop'

  steps:
  - name: Download Artifacts
    uses: actions/download-artifact@v3
    with:
      name: service-package

  - name: Deploy to Staging
    run: |
      platform-cli deploy staging package.tar.gz \
        --wait-healthy \
        --timeout 5m

  - name: Run Smoke Tests
    run: |
      platform-cli test smoke --environment staging

deploy-production:
  needs: test
  runs-on: ubuntu-latest
  if: github.ref == 'refs/heads/main'

  steps:
  - name: Download Artifacts
    uses: actions/download-artifact@v3
    with:
      name: service-package

  - name: Deploy to Production
    run: |
      platform-cli deploy production package.tar.gz \
        --strategy blue-green \
        --wait-healthy \
        --timeout 10m

  - name: Run Production Tests
    run: |
      platform-cli test smoke --environment production
      platform-cli test performance --environment production --quick
```

Deployment Strategies

```bash
# Blue-green deployment
platform-cli deploy production --strategy blue-green

# Canary deployment
platform-cli deploy production --strategy canary --percentage 10
```

Best Practices

Code Organization

```
MyService/
├── src/Services/MyService.cs
├── schemas/MyService.openrpc.json
├── tests/MyServiceTests.cs
└── platform.yaml
```

Error Handling

```csharp
try
{
    return await ProcessRequest(request);
}
catch (ValidationException ex)
{
    return new ErrorResponse { Code = "VALIDATION_ERROR", Message = ex.Message };
}
catch (Exception ex)
{
    _logger.LogError(ex, "Unexpected error");
    return new ErrorResponse { Code = "SYSTEM_ERROR", Message = "An error occurred" };
}
```

Input Validation

csharp

```csharp
// Schema-based validation is automatic, but add business validation
public async Task<CreateUserResponse> CreateUser(CreateUserRequest userData)
{
    // Business rule validation
    if (await IsEmailDomainBlocked(userData.Email))
    {
        throw new ValidationException("Email domain is not allowed");
    }

    if (!await IsDepartmentValid(userData.Department))
    {
        throw new ValidationException("Invalid department specified");
    }

    // Proceed with user creation...
}

private async Task<bool> IsEmailDomainBlocked(string email)
{
    var domain = email.Split('@').LastOrDefault();
    var blockedDomains = await _dataStore.Query("blocked_domains",
        Query.Where("domain", domain));
    return blockedDomains.Any();
}
```

Testing Best Practices

Test Organization:

csharp

```csharp
// Organize tests by feature/scenario
public class UserCreationTests : ServiceTestBase<UserService>
{
    [Fact]
    public async Task CreateUser_ValidData_ReturnsSuccess() { }

    [Fact]
    public async Task CreateUser_DuplicateEmail_ReturnsError() { }

    [Theory]
    [InlineData("")]
    [InlineData("invalid-email")]
    public async Task CreateUser_InvalidEmail_ReturnsValidationError(string email) { }
}

public class UserRetrievalTests : ServiceTestBase<UserService>
{
    [Fact]
    public async Task GetUser_ExistingUser_ReturnsUser() { }

    [Fact]
    public async Task GetUser_NonExistentUser_ReturnsNotFound() { }

    [Fact]
    public async Task GetUser_UnauthorizedAccess_ReturnsUnauthorized() { }
}

// Test data builders for consistency
public class UserTestDataBuilder
{
    private string _email = "test@example.com";
    private string _name = "Test User";
    private string _department = "Engineering";

    public UserTestDataBuilder WithEmail(string email)
    {
        _email = email;
        return this;
    }

    public UserTestDataBuilder WithName(string name)
    {
        _name = name;
```

```
        return this;
    }

    public UserTestDataBuilder WithDepartment(string department)
    {
        _department = department;
        return this;
    }

    public CreateUserRequest Build()
    {
        return new CreateUserRequest
        {
            Email = _email,
            Name = _name,
            Department = _department
        };
    }
}
```

Development Configuration

Service Configuration (platform.yaml):

yaml

```yaml
service:
 name: UserService
 version: "1.0.0"
 language: csharp

 dependencies:
  - postgresql
  - redis

 environment:
  development:
   database:
    connectionString: "postgresql://localhost:5432/userservice_dev"
    migrations: auto

   messaging:
    broker: "mqtt://localhost:1883"
    topics:
     - "service/UserManagement"
     - "events/user/*"

   monitoring:
    tracing: enabled
    metrics: enabled
    logging: debug

  production:
   database:
    connectionString: "${DATABASE_CONNECTION_STRING}"
    migrations: manual

   messaging:
    broker: "${MQTT_BROKER_URL}"
    topics:
     - "service/UserManagement"
     - "events/user/*"

   monitoring:
    tracing: enabled
    metrics: enabled
    logging: info

 resources:
```

```
  development:
    memory: "512Mi"
    cpu: "0.5"
  production:
    memory: "2Gi"
    cpu: "2"
    replicas: 3
```

Troubleshooting

Common Issues and Solutions

Service won't start:

```bash
# Check service logs
platform-cli logs --service UserService --tail 50

# Check service status
platform-cli debug status UserService

# Validate configuration
platform-cli validate config --service UserService

# Common outputs:
# ❌ Database connection failed: Connection refused
# ❌ MQTT broker unreachable: Network timeout
# ❌ Schema validation failed: Invalid JSON format
```

Slow deployments:

bash

```
# Warm up build cache
platform-cli cache warm --all-runtimes

# Check build performance
platform-cli debug build-performance

# Optimize container layers
platform-cli optimize dockerfile --service UserService

# Output analysis:
# 🐌 Slow step: dotnet restore (45s)
# 💡 Suggestion: Add .dockerignore to reduce context size
# 💡 Suggestion: Use multi-stage build for dependencies
```

MQTT connection issues:

bash

```
# Check MQTT broker status
platform-cli debug mqtt status

# Test MQTT connectivity
platform-cli debug mqtt connectivity --host localhost --port 1883

# Monitor message flow
platform-cli debug mqtt trace --follow

# Example diagnostics:
# ✅ MQTT broker: Running (localhost:1883)
# ❌ Client connection: Failed authentication
# 🔍 Active topics: 23 (service/*, events/*)
```

Database problems:

bash

```
# Test database connectivity
platform-cli debug database connectivity

# Check database permissions
platform-cli debug database permissions --user UserService

# Validate schema migrations
platform-cli debug database schema --service UserService

# Reset development database
platform-cli dev database reset --confirm

# Output examples:
# ✅ PostgreSQL: Connected (localhost:5432)
# ❌ Table 'users': Missing column 'created_at'
# 🔧 Migration pending: 2024_01_15_add_department_field
```

Performance issues:

bash

```
# Monitor resource usage
platform-cli monitor resources --service UserService --live

# Profile memory usage
platform-cli profile memory --service UserService --duration 60s

# Analyze slow queries
platform-cli debug database slow-queries --service UserService

# Check message queue backlog
platform-cli debug messaging backlog

# Sample output:
# 📊 CPU: 85% (threshold: 80%)
# 📊 Memory: 1.2Gi/2Gi (60%)
# 🐌 Slow query: SELECT * FROM users (avg: 250ms)
# 📬 Queue backlog: 150 messages
```

Schema validation errors:

```bash
# Validate all schemas
platform-cli validate schemas --strict

# Check schema compatibility
platform-cli validate compatibility --from v1.0.0 --to v1.1.0

# Generate schema documentation
platform-cli generate docs --schemas

# Fix common schema issues
platform-cli fix schemas --auto-correct

# Example errors:
# ❌ UserService.openrpc.json: Invalid email format regex
# ❌ Breaking change: Removed required field 'name'
# ⚠️  Deprecated: Field 'legacy_id' should be removed
```

Debug Dashboard Usage

Access the Debug Dashboard:

```bash
# Open debug dashboard
platform-cli debug dashboard

# Dashboard available at: http://localhost:8080
```

Dashboard Features:

- **Service Health**: Real-time status of all services
- **Message Flow**: Visual representation of service communication
- **Performance Metrics**: CPU, memory, and response time graphs
- **Trace Explorer**: Distributed tracing visualization
- **Log Aggregation**: Centralized log viewing with filtering
- **Schema Browser**: Interactive API documentation

Advanced Debugging Techniques

Remote Debugging Setup:

```csharp
// Add to Program.cs for development
#if DEBUG
if (Environment.GetEnvironmentVariable("PLATFORM_ENV") == "development")
{
    app.Services.GetRequiredService<ILogger<Program>>()
        .LogInformation("🐛 Remote debugging enabled on port 5005");

    // Enable hot reload and debugging
    app.UseHotReload();
    app.UseRemoteDebugging(port: 5005);
}
#endif
```

Live Code Changes:

```bash
# Enable live coding mode
platform-cli dev start --live-coding

# Make changes to code files - they're applied immediately
# No need to restart or redeploy during development
```

Message Tracing:

```bash
# Trace specific message flows
platform-cli trace message --pattern "user.created" --follow

# Trace cross-service calls
platform-cli trace cross-service --from UserService --to OrderService

# Output example:
# 📨 Message: user.created
#    ├─UserService →events/user/created (2ms)
#    ├─OrderService ←events/user/created (5ms)
#    ├─NotificationService←events/user/created (3ms)
#    └─AuditService ←events/user/created (1ms)
```

Advanced Development Patterns

Event-Driven Architecture

Publishing Events:

csharp

```
public async Task<CreateUserResponse> CreateUser(CreateUserRequest userData)
{
    // Create user logic...
    var userId = await _dataStore.Insert("users", userData);

    // Publish event for other services
    await _messaging.Publish("events/user/created", new UserCreatedEvent
    {
        UserId = userId,
        Email = userData.Email,
        Name = userData.Name,
        CreatedAt = DateTime.UtcNow
    });

    return new CreateUserResponse { UserId = userId, Success = true };
}
```

Consuming Events:

csharp

```
[EventHandler("events/user/created")]
public async Task HandleUserCreated(UserCreatedEvent userEvent)
{
    // Send welcome email
    await _emailService.SendWelcomeEmail(userEvent.Email, userEvent.Name);

    // Create user profile
    await _profileService.CreateProfile(userEvent.UserId);

    // Log user creation
    _logger.LogInformation("User {UserId} created successfully", userEvent.UserId);
}
```

Saga Pattern Implementation

Order Processing Saga:

csharp

```csharp
[Saga("OrderProcessing")]
public class OrderProcessingSaga
{
  [SagaStart]
  public async Task Handle(OrderCreatedEvent orderEvent)
  {
    // Step 1: Reserve inventory
    await SendCommand(new ReserveInventoryCommand
    {
      OrderId = orderEvent.OrderId,
      ProductId = orderEvent.ProductId,
      Quantity = orderEvent.Quantity
    });
  }

  public async Task Handle(InventoryReservedEvent inventoryEvent)
  {
    // Step 2: Process payment
    await SendCommand(new ProcessPaymentCommand
    {
      OrderId = inventoryEvent.OrderId,
      Amount = inventoryEvent.TotalAmount
    });
  }

  public async Task Handle(PaymentProcessedEvent paymentEvent)
  {
    // Step 3: Ship order
    await SendCommand(new ShipOrderCommand
    {
      OrderId = paymentEvent.OrderId
    });
  }

  // Compensation handlers for failures
  public async Task Handle(PaymentFailedEvent paymentFailedEvent)
  {
    await SendCommand(new ReleaseInventoryCommand
    {
      OrderId = paymentFailedEvent.OrderId
    });
  }
}
```

Conclusion

The Service Function Architecture development environment provides a comprehensive framework that transforms how teams build, test, and deploy distributed applications:

Key Benefits Delivered

Development Excellence:

- **Rapid Development**: 3-5 second deployment cycles with full container isolation
- **Production Parity**: Identical development and production environments eliminate deployment surprises
- **Cross-Language Support**: Seamless integration across programming languages with automatic schema generation
- **Zero-Configuration Setup**: Complete development environment ready in minutes

Testing Superiority:

- **Real Platform Testing**: No complex mocking required - test against actual infrastructure
- **Comprehensive Coverage**: Unit, integration, performance, and end-to-end testing with a unified framework
- **Automatic Test Generation**: Schema-driven test creation reduces boilerplate
- **Performance Validation**: Built-in load testing and resource monitoring

Operational Confidence:

- **Advanced Debugging**: Distributed tracing and structured logging provide deep visibility
- **Multi-Service Development**: Complex application development across service boundaries
- **CI/CD Integration**: Seamless pipeline integration with automated testing and deployment
- **Monitoring and Observability**: Real-time dashboard and comprehensive metrics

Developer Experience:

- **Schema-First Development**: OpenRPC-driven development eliminates API maintenance overhead
- **Live Feedback**: Real-time log streaming, performance metrics, and hot reload capabilities
- **Unified Tooling**: Single CLI for all development, testing, and deployment operations
- **Documentation Integration**: Automatic API documentation generation from schemas

Architecture Advantages

This framework enables teams to focus on business logic implementation while the platform handles infrastructure complexity, security, scalability, and operational concerns. The result is faster delivery of high-quality applications with enterprise-grade reliability.

The combination of rapid development cycles, comprehensive testing capabilities, and production-identical environments creates a development experience that significantly reduces time-to-market while maintaining code quality and system reliability.

Getting Started

To begin using this development framework:

1. **Install the Platform**: `curl -sSL https://install.platform.dev/dev | bash`
2. **Create Your Service**: `platform-cli create service MyService --language csharp`
3. **Start Developing**: `platform-cli dev start --watch`

The platform will guide you through the development process with intelligent defaults, comprehensive documentation, and real-time feedback to ensure your success.