Master Test Strategy

A common approach across all projects

Document Purpose and Scope **3**
Testing Basics **3**
Strata of Test Types 3
Test Plan 4
Proposed Automated Tests 4
Summary 4
Meshery Docs 5
Meshery Server and Meshery UI 5
Meshery CLI/mesheryctl 6
Meshery Adapters 7
Meshery Install Artifacts 7
Other Testing Considerations 7
Testing Methods 8
UI Testing 8
Pros: 8
Cons: 8
Function/Module Level Testing 9
Pros 9
Cons 9
Criteria for making a release 9
Beta 9
Rc 9

Document Purpose and Scope

The purpose of this document is to layout the master test strategy across the Layer5 projects. Layer5's master test strategy focuses on the approach to testing, the strata of test types, when they are run, for what purpose, and so on, while a given project's test plan focuses on specific test cases and their status.

Purpose of Testing

- To ensure that the intent of the application is met.
- To enhance confidence in the results put out by the application.
- To ensure that changes/enhancements made have not broken the application.

Scope

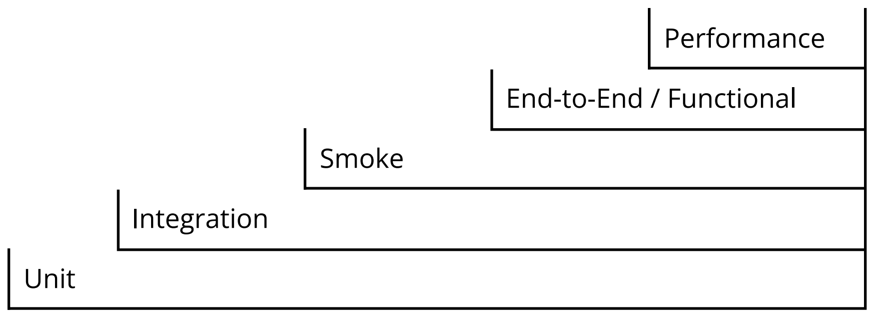
All Layer5 projects are in scope except prototypes, those that are in the very early stages of conception and such candidates.

Testing Basics

The following are the basic aspects to consider in setting up a test strategy:

- Integration Testing: An end-to-end test which will run through all the user requirements. This could be a test that can be run at a major release.
- Unit testing: A test that is conducted for a unit of the application such as a function/procedure/module/functionality.
- Regression Testing: A series of tests run to ensure that existing functionality was not broken as a result of introducing changes.

Strata of Test Types



1. **Performance** - quantitative; high load; soak over extended period; or spike.
2. **User Acceptance Testing** - qualitative; manually performed by users.
3. **End-to-End / Functional** - broad coverage; happy path functionality.
4. **Smoke** - basic; kick the tires; does it stand up or does it fall over?

5. **Integration** - inter-component; great for negative tests (missing prerequisites).
6. **Unit** - intra-component - narrow; specific and edge case tests.

Test Plan

See Automated Tests for the set of current tests.

Proposed Automated Tests

The test names defined here correspond to `workflow name/job name` in the GitHub workflows.

Summary

Workflow	Job	Area	Dependency
Meshery Docs CI	Build and Preview Site	Meshery Docs	None
Meshery Server Build	Go Lint Check	Meshery Server	None
Meshery Server Build	Build Meshery Server	Meshery Server	None
Meshery UI Build	Build Meshery UI	Meshery UI	None
Meshery Server and UI CI	Run Meshery Server Unit Tests	Meshery Server	Meshery Server Build/Build Meshery Server
Meshery Server and UI CI	Run Meshery Server Integration Tests	Meshery Server	Meshery Server Build/Build Meshery Server
Meshery Server and UI CI	Run Meshery UI Integration Tests	Meshery UI	Meshery UI Build/Build Meshery UI
Meshery Server and UI CI	Run Meshery Smoke Test	Meshery UI, Meshery Server	Meshery UI Build/Build Meshery UI, Meshery Server and UI CI/Run Meshery UI Integration Tests
Meshery Server and UI CI	Run Meshery End-to-End Tests	Meshery UI, Meshery Server	Meshery UI Build/Build Meshery UI, Meshery Server and UI CI/Run Meshery UI Integration Tests
Meshery Server and UI CI	Build Meshery Docker Image	Meshery Server, Meshery UI, Dockerfile	None
Meshery Server and UI CI	Generate REST Docs	Meshery Server, Meshery Docs	None

Meshery Server and UI CI	Generate GraphQL Docs	Meshery Server, Meshery Docs	None
mesheryctl Build	Go Lint Check	mesheryctl	None
mesheryctl Build	Build mesheryctl	mesheryctl	None
mesheryctl CI	Run mesheryctl Unit Tests	mesheryctl	None
mesheryctl CI	Run mesheryctl Integration Tests	mesheryctl	None
mesheryctl CI	Run mesheryctl Smoke Tests	mesheryctl	mesheryctl
mesheryctl CI	Run mesheryctl End-to-End Tests	mesheryctl, Meshery Server	mesheryctl Build/Build mesheryctl, mesheryctl CI/ Run mesheryctl Smoke Tests
Publish Helm Charts	Validate Helm Charts	Helm charts	None

Meshery Docs

Changes made directly to Meshery docs trigger a deploy preview. Failure in builds fails the check. Quality of changes verified manually.

Test Name: Meshery Docs CI/Build and Preview Site
Kind: Smoke
Test Summary: Builds and shows a preview of the docs site
Runs On: Pull requests to Master with changes to `docs/**`

Test Name: Meshery Server and UI CI/Generate REST Docs
Kind: Pseudo-Integration
Test Summary: Generates Swagger docs from code
Runs On: Pull requests to Master with changes to `handlers/**`

Test Name: Meshery Server and UI CI/Generate GraphQL Docs
Kind: Pseudo-Integration
Test Summary: Generates GraphQL docs from GraphQL schema
Runs On: Pull requests to Master with changes to `internal/graphql/schema/schema.graphql`

Meshery Server and Meshery UI

Runs on all changes in Meshery except for changes to Meshery docs and mesheryctl.

These tests are run in the same order as shown below.

Test Name: Meshery Server Build/Go Lint Check
Kind: Static
Test Summary: Runs a Golang lint check with golangci-lint
Runs On: Pull requests to Master, workflow dispatch

Test Name: Meshery Server Build/Build Meshery Server
Kind: Smoke
Test Summary: Builds Meshery Server
Runs On: Pull requests to Master, workflow dispatch

Test Name: Meshery UI Build/Build Meshery UI
Kind: Smoke
Test Summary: Builds Meshery UI
Runs On: Pull requests to Master, workflow dispatch

Test Name: Meshery Server and UI CI/Run Meshery Server Unit Tests
Kind: Unit
Test Summary: Runs unit tests defined for Meshery Server
Runs On: Pull requests to Master, workflow dispatch
Depends On/Needs: Meshery Server Build/Build Meshery Server

Test Name: Meshery Server and UI CI/Run Meshery Server Integration Tests
Kind: Integration
Test Summary: Runs integration tests defined for Meshery Server
Runs On: Pull requests to Master, workflow dispatch
Depends On/Needs: Meshery Server Build/Build Meshery Server

Test Name: Meshery Server and UI CI/Run Meshery UI Integration Tests
Kind: Integration
Test Summary: Runs Cypress integration tests defined for Meshery UI
Runs On: Pull requests to Master, workflow dispatch
Depends On/Needs: Meshery UI Build/Build Meshery UI

Test Name: Meshery Server and UI CI/Run Meshery Smoke Test
Kind: Smoke
Test Summary: Runs a smoke test with locally build Meshery Server and Meshery UI with any one of the adapters
Runs On: Pull requests to Master, workflow dispatch
Depends On/Needs: Meshery Server Build/Build Meshery Server, Meshery UI Build/Build Meshery UI

Test Name: Meshery Server and UI CI/Run Meshery End-to-End Tests
Kind: End-to-End
Test Summary: Runs Cypress End-to-End tests defined for Meshery UI
Runs On: Pull requests to Master, workflow dispatch
Depends On/Needs: Meshery Server Build/Build Meshery Server, Meshery UI Build/Build Meshery UI

Meshery CLI/mesheryctl

Runs on all changes to `mesheryctl/**`.

Test Name: mesheryctl Build/Go Lint Check
Kind: Smoke
Test Summary: Runs a Golang lint check with golangci-lint
Runs On: Pull requests to Master, workflow dispatch

Test Name: mesheryctl Build/Build mesheryctl
Kind: Smoke
Test Summary: Builds mesheryctl
Runs On: Pull requests to Master, workflow dispatch

Test Name: mesheryctl CI/Run mesheryctl Unit Tests
Kind: Unit
Test Summary: Runs unit tests defined for mesheryctl
Runs On: Pull requests to Master, workflow dispatch

Test Name: mesheryctl CI/Run mesheryctl Integration Tests
Kind: Unit
Test Summary: Runs integration tests defined for mesheryctl
Runs On: Pull requests to Master, workflow dispatch

Test Name: mesheryctl CI/Run mesheryctl Smoke Tests
Kind: Smoke
Test Summary: Runs smoke tests with
Runs On: Pull requests to Master, workflow dispatch
Depends On/Needs: mesheryctl Build/Build mesheryctl

Test Name: mesheryctl CI/Run mesheryctl End-to-End Tests
Kind: End-to-End
Test Summary: Runs End-to-End tests with mesheryctl
Runs On: Pull requests to Master, workflow dispatch
Depends On/Needs: mesheryctl Build/Build mesheryctl

Meshery Adapters

Runs on all changes to Meshery Adapters (in each of the adapter repos).

TBD.

Meshery Install Artifacts

Test Name: Meshery Server and UI CI/Build Meshery Docker Image
Kind: Smoke
Test Summary: Builds Meshery Docker image
Runs On: Pull requests to Master, workflow dispatch

Test Name: Publish Helm Charts/Validate Helm Charts
Kind: Smoke
Test Summary: Uses "Helm Lint" to validate the Helm charts
Runs On: Pull requests to Master, workflow dispatch with changes to `install/kubernetes/**`

Other Testing Considerations

Here are some other aspects of testing, some of which are taking on more importance especially because of the greater penetration that applications have today. Quite a few of these are design issues. However, they do need to be addressed in testing. Also, not all tests are pass/fail such as performance or scalability. A reliable and objective measure(s) need to be derived to determine if an application has performed satisfactorily.

- **Accessibility testing**: Ensure that all people are capable of accessing and operating the application. This is especially true if the application is available on the internet or is available as a mobile app. A lot depends on the intent of the application creator, however, general availability deems this important. This should take into account people who may be handicapped in some way, or of a different culture. The nature of the application may preclude some of these requirements.
- **Installability**: This goes beyond just operating system and platform considerations. It is true that as more and more applications become mobile, this may not seem an issue. This goes to dependencies, and clashes with them if they already exist, and other data/metadata/configuration that may be required to run the application successfully.
- **Backward and forward compatibility**: Of course, backward and forward compatibility cannot be guaranteed as that impedes the ability to put in new functionality. But it is important to not leave users completely stranded. This needs to be tested.
- **Performance testing**: A measure of how fast the application instance handles a greater load and the ability of its dependents/components to respond to that load.
- **Scalability**: A measure of how well the application musters more resources to handle an increasing number of users and/or load.
- **Security**: How secure is the application? While it is not possible to determine the entire set of ways in which an application can be infiltrated (as there is a lot of innovation going on in that area too), the application must be tested for well-known vulnerabilities.

Testing Methods

UI Testing

This is where the user-interface is subjected to all the possible interactions and a record is made of any deviations that may occur from the expected result. Examples of such frameworks are Selenium and Cypress amongst a myriad of others.

Pros:

- Quick way to ensure that the basic functionality is met.
- Capable of being automated to give a pass/fail verdict.
- Can be compartmentalized to run tests selectively as required.

Cons:

- Cannot identify defects in individual program elements
- Cannot effectively validate performance requirements.
- Cannot effectively scalability.
- Can measure conformance to requirements from an individual perspective only.
- Cannot detect security vulnerabilities.
- Needs to be constantly updated as changes are made to the existing software. There is no close link between development and the actual user interface as the development may be buried in the deepest layers of the software.
- Cannot be used to test functionality in CLI software such as mesheryctl.

Function/Module Level Testing

This is where a parallel project is created to mirror the individual components of the application. For each function/procedure, a test function(s) is/are created to validate the functioning of that function/procedure by calling that function/procedure with the required parameters and asserting that the output is valid.

Pros

- Closely couples application changes to the tests.
- Capable of identifying the exact function/procedure that caused the error.
- Capable of being automated.
- Suitable for applications such as mesheryctl.

Cons

- Is not very suitable for the UI.
- Not suitable for identifying security vulnerabilities.
- Not suitable for identifying scalability deficiencies.

Adds extra development effort.

Criteria for making a release

Beta

Ensure that the major introductions have had their functionality validated.

Rc

Ensure that the major introductions for that release have been validated and that the application is close to the previous stable version in its functionality.