BETA TEST PLAN - Wormhole

Written by Axel Denis, Julian Scott, Ludovic de Chavagnac and Arthur Aillet

1. Core Functionalities for Beta Version

Feature Name	Description	Priority (High/Medium/Low)	Changes Since Tech3
Basic Local Interactions	Read, Write, Move, Rename, Delete files and Folders. All common user interactions must be implemented	High	Improved and expanded
Docker Image	Replicable environment	Medium	Planned
Configuration Files	Wormhole is configurable by configuration file.	Medium	Not Fixed
Stability (Almost)	Does not generate spontaneous errors. Can handle invalid input with safe failures	High	Improved
Linux Support	Complete support for all majors Linux systems (Debian, Arch, etc.).	High	Validated
Windows Support	Support of the windows system post windows 10	High	Added
Mac Support (lack testing)	Functional on mac systems.	Low	Theorhetically added
Complete User Documentation	The user and developper documentation is clear and easy to use.	High	Expanded
Redudancy	Basic redundancy with a minimum of 3 pods.	High	Update redudancy (2 pods to a theoretically unlimited number of pods)
Compliant Error Handling	Software interacting with Wormhole may respond accordingly	Medium	Fixed
Cache Handling	Optimized cache management for improved performance.	Medium	Planned
CLI Interface	Optimized cache management for improved performance.	Medium	Planned
Error Resilience	Non-fatal errors can be recovered from.	Hight	Improved
Fancy optimization strategies	Traffic optimization depending on network speed or storage available	None	Aborted feature
Pod specialisation	Pods with different roles and features	None	Aborted feature

2. Beta Testing Scenarios

2.1 User Roles

Role Name	Description
Windows user	Uses a windows computer without necessarily much experience with computers
Home server user	Manages a home server with a large archive of files that represents a real-world scenario of data storage and distribution
Professionnal Developper	Intergrates software within their company's ecosystem. Has a lot of experience with different APIs, capable of giving feedback on it, and knows industry best-practises and corporate interests

Role Name	Description
Open source Contributor	Contributes to open source projects, has no key specialisations but would like to contribute on Wormhole
Mac user	Uses a Mac computer, may have high standards for interface, and would give important feedback.

2.2 Test Scenarios

For each core functionality, provide detailed test scenarios.

Scenario 0: First Installation

- **Role Involved:** For every role
- **Objective:** Verify that the new user onboarding process is simple and clear.
- **Prerequisites:** None
- **Test Steps:**
    1. Get access to the wormhole documentation and installation page
    2. Create a new network and pod using the CLI
    3. use the CLI to join the network with a second pod on the same machine
    4. (For advanced users) Create a third instance on the same local area network, and connect it to the others
    5. Test that all instances are properly connected by adding a blank file to the network
- **Expected Outcome:** The user doesn't need to consult any external resources for installation, and has a functional network

Scenario 1: Basic Local Interactions

- **Role Involved:** For every role
- **Objective:** Interact with the filesystem (read, write, move, rename, delete, ...) a file or folder.
- **Prerequisites:** Installing Wormhole With a single instance
- **Test Steps:**
    1. Create a file
    2. Create a folder
    3. Write to a file
    4. Append to a file
    5. Read a file
    6. Create files in a folder
    7. Create folders in a folder
    8. Move a file into a folder
    9. Move an empty folder into an other folder
    10. Move a non-empty folder into an other folder
    11. Move a file from an external filesystem to the wormole drive
    12. Move a file from the wormhole drive to an external filesystem
    13. Move a non-empty folder from an external filesystem to the wormhole drive
    14. Move a non-empty folder from the wormhole drive to an external filesystem
    15. Rename a file
    16. Rename a folder
    17. Delete a file
    18. Delete an empty folder
    19. Delete a non-empty folder recursively
    20. List the content of a folder
    21. Look at properties for a file or folder
    22. Change permissions of file, retry previous relevant steps
    23. Change permissions of a folder, retry previous relevant steps
- **Expected Outcome:** No errors surprise the user. Actions that would be illegal on any filesystem return the correct error

Scenario 2: Complex interactions

- **Role Involved:** For every role
- **Objective:** Run an app with its files hosted on an other Wormhole instance
- **Prerequisites:** Installing Wormhole with two instances on the network
- **Test Steps:**
    1. Install Gimp on the first host
    2. Open the app on the second host
    3. Record the startup time
    4. Create a simple image using the software
    5. Foward the wormhole logs in case of any errors
- **Expected Outcome:** The experience should feel seemless

Scenario 3: Docker Image

- **Role Involved:** Professionnal developper, home server user
- **Objective:** Testing wormhole with a docker image
- **Prerequisites:** Installing Wormhole and Docker
- **Test Steps:**
    1. Clone or download the Wormhole repository then build and run the provided simple Wormhole Docker image.
    2. Launch the Docker container with the appropriate command.
    3. Check that Wormhole mounts correctly by consulting the logs or the container status.
    4. Launch a second docker container on another machine, and check that both instances are connected to each other.
    5. Performs tests similar to the Scenario 2 (Basic local interactions)
    6. Check for errors on the utilisation of this third party app.
- **Expected Outcome:** The Docker image runs Wormhole correctly, and all basic file system operations run smoothly. Any errors are easily viewed through the docker container. Third party apps can function using the container.

Scenario 4: Configuration Files

- **Role Involved:** For every role
- **Objective:** Create a configuration file based on the documentation and use it.
- **Prerequisites:** Installing Wormhole
- **Test Steps:**
    1. Use the Wormhole CLI to generate a default configuration file.
    2. Modify one setting (e.g. mount point or redundancy parameters).
    3. Start Wormhole with the modified configuration and check that the changes are in place.
    4. Modify another setting (e.g. cache settings).
    5. Reload configuration using the CLI without restarting the wormhole service.
    6. Check that the new configuration has been applied
- **Expected Outcome:** Documentation is clear and easy to use. The wormhole drive follows the configuration correctly.

Scenario 5: Stability

- **Role Involved:** For every role
- **Objective:** Test Wormhole's stability under various operations without crashes or critical bugs.
- **Prerequisites:** Installing Wormhole and configuring a basic instance
- **Test Steps:**
    1. Perform a rapid series of operations: create several files, write, read, delete, rename.
    2. Attempt operations that are expected to fail (e.g. deleting a non-existent file).
    3. Run Wormhole for 24 hours with periodic operations.
    4. Upload and download large files (e.g. 5GB).
    5. Monitor logs for errors or warnings.
    6. Interrupt the Wormhole instance (e.g. sudden stop) then restart it and check if the drive behaves as expected.
    7. Interrupt the Wormhole instance and check if the owned files are still available without wormhole running
- **Expected Outcome:** Wormhole manages all operations without crashing, logs errors appropriately and maintains data integrity.

Scenario 6: Linux Support

- **Role Involved:** Home server user
- **Objective:** Testing if Wormhole works on GNU/Linux systems
- **Prerequisites:** Installing Wormhole
- **Test Steps:**
    1. Install Wormhole on any major Linux distribution.
    2. Execute common operations on a filesystem (creating, reading and editing files).
    3. Use and test advanced features with configuration files like redundancy.
    4. Test linux specific features: xattr editions
    5. Use this system accross many of your servers, with external apps that could benefit from large distributed storage (ex. storage of pictures like Immich), and see if they are able to operate normally.
- **Expected Outcome:** Wormhole installs and runs correctly on several Linux distributions, and can be used by other programs that require file storage.

Scenario 7: Mac Support

- **Role Involved:** Mac user
- **Objective:** Testing if Wormhole works on Mac systems.
- **Prerequisites:** Installing Wormhole
- **Test Steps:**
    1. Install Wormhole on a macOS system.
    2. Perform basic operations: create, read, write and delete files and folders.
    3. Check macOS-specific issues (e.g. permissions, Finder integration).
    4. Test the user interface to ensure it is intuitive for Mac users.
- **Expected Outcome:** Wormhole runs on macOS with the correct basic operations, although limitations may exist due to not currently being officially supported.

Scenario 8: Windows Support

- **Role Involved:** Windows user
- **Objective:** Testing if Wormhole works on Windows systems.
- **Prerequisites:** Installing Wormhole
- **Test Steps:**
    1. Install Wormhole on a Windows 10 or windows 11 system.
    2. Perform basic operations: create, read, write, delete files and folders.
    3. Test Windows-specific integrations (File explorer).
    4. Identify unimplemented or buggy features (following the github issues).
- **Expected Outcome:** Wormhole installs and runs smoothly on Windows, with most features operational.

Scenario 9: Redudancy

- **Role Involved:** For every role
- **Objective:** Testing the redundancy of the system.
- **Prerequisites:** Installing Wormhole on 3 different computers: A, B, and C
- **Test Steps:**
    1. Create a Wormhole Network by creating an instance on computer *A*.
    2. Join the Network on each of the other machines with the Wormhole CLI
    3. Create a different file on each machine.
    4. Check that all the files are available on all machines.
    5. Shutdown computer *A*.
    6. Check that the files remain available on the remaining computers: they must have automatically retrieved *A*'s file.
    7. Modify computer *A*'s file on machine *B*.
    8. Restart computer *A* and rejoin the network.
    9. Check that the modified file is updated on machine *A*.
- **Expected Outcome:** The system maintains file availability and consistency, even in the event of a node failure, and changes are propagated correctly.

Scenario 10: Complete User Documentation

- **Role Involved:** For every role
- **Objective:** Check if the documentation is clear and easy to use for a new user.
- **Prerequisites:** Previous tasks
- **Test Steps:**
    1. During the scenario 1, the user read the documentation to install Wormhole.
    2. During the scenario 4, the user read the documentation to configure Wormhole.
    3. After experiencing in first hand the documentation, the user can identify sections that felt incomplete or unclear.
    4. Suggest improvements or additions to the content.
- **Expected Outcome:** If the documentation is clear and easy to use the user should be able to install, use, and configure Wormhole without any help.

Scenario 11: Clean Error Handling

- **Role Involved:** For every role
- **Objective:** Testing the error handling of the system are complete and understandable by the user and interfacing software.
- **Prerequisites:** Wormhole network of multiple instances
- **Test Steps:**
    1. Attempt an operation that should fail (e.g. write to a read-only file or access a non-existent file).
    2. Observe the error message or provided by Wormhole or the file manager interfacing with wormhole.
    3. Check that the message is clear and correct.
    4. Test error handling in different contexts.
    5. Check that errors are logged for debugging purposes.
- **Expected Outcome:** Wormhole provides clear and useable error messages that the user can use for troubleshooting.

Senario 12: CLI Interface

- **Role Involved:** For every role
- **Objective:** Verify the CLI works has expected
- **Prerequisites:** Working wormhole service
- **Test Steps:**
    1. Use the CLI to generate a template pod configuration
    2. Use the CLI to create a pod (new network) (with the template configuration)
    3. Check that the created pod is working properly
    4. Use the CLI to stop that pod.
    5. Use the CLI to join a network (new pod)
    6. Check that the pod is working correctly (access to network shared files)
    7. Stop the pod using the cli, then restart it
    8. Verify that you can still access the files.
    9. Use the "inspect" command to have a basic report of the pod's state
- **Expected Outcome:** The cli must be able to create a network, operate it and show the network state without failure

Senario 13: Error Resilience

- **Role Involved:** For Professionnal developper, home server user
- **Objective:** Verify that fatal errors are handled properly on a network scale
- **Prerequisites:** Wormhole network of multiple instances
- **Test Steps:**
    1. Create a network with the following needs:
        1. At least 3 pods
        2. Redundancy at least 1
    2. Create files on the network
    3. Ensure created files are accessible on all pods
    4. Pkill one of the pods
    5. Check all files are present, can be read and modified.
- **Expected Outcome:** All the other network instances must continue working properly even if an instance has a fatal issue. No files should be lost.

---

3. Success Criteria

The following criteria will be used to determine the success of the beta version.

Criterion	Description	Threshold for Success
Stability	No major crashes or critical bugs	No crash reported
Usability	End users can use wormhole like any cloud storage	80% positive feedback from testers
Performance	95% of individual files operation are completed in less than 10 seconds	95% of files processed within 2 seconds
Accuracy	The wormhole drive behaves like any drive in the eyes of the system	90% correctness in drive tests

---

4. Known Issues & Limitations

[List any known bugs, incomplete features, or limitations that testers should be aware of.]

Issue	Description	Impact	Planned Fix? (Yes/No)
Window support	Incomplete support of the windows system	High	Yes
Documentation	Incomplete documentation for user or developper	Medium	Yes
... To expand with the current features when sharing the beta test plan with testers.			

5. Conclusion

This **Beta Test Plan** for Wormhole describes the essential steps for testing core functionality across different user roles and scenarios. By involving a variety of testers (Mac users, Windows users, home servers, engineers and professional developers), we aim to gather comprehensive feedback on usability, stability and performance. The success criteria defined will enable us to assess whether Wormhole is ready for a wider release. Resolving known issues and limitations during the beta phase will be crucial to delivering a robust, user-friendly product. We hope that the insights gained from these tests will guide final adjustments and ensure that Wormhole meets the high expectations of our users.