

Test Plan and Results

Team Members: Kyle Cullion, Michael Keenan, Vivek Kunapareddy, Zackery Steck

Overall Test Plan

Our approach to testing will contain two main phases. First, we will test the system components individually, using simulated data, covering a variety of normal and abnormal conditions. This will help us verify coverage of edge cases. We will create various test cases for all aspects of the system. These will test both normal and abnormal cases. For the second phase, we will test all system components in a production environment, using multiple personal machines and AWS instances. This will further test the system functionality and provide stress testing, to make sure that it can handle the load of a production environment.

Test Case Descriptions

MS 1.1 Agent Registration Test

MS 1.2 Verify the linking of agents to customer accounts is working properly

MS 1.3 Install an agent on an available server, and attempt to link it to a test account to verify functionality

MS 1.4 Inputs: The new agent installation on the server

MS 1.5 Outputs: Visual verification that the agent is sending back monitoring data to the test account when requested

MS 1.6 Normal

MS 1.7 Whitebox

MS 1.8 Functional

MS 1.9 Integration

MS 2.1 Hourly Check-in on Master Server

MS 2.2 This test verifies the core functionality of the server monitoring process

MS 2.3 Ping the listening server, verify successful ping returned

MS 2.4 Inputs: Pinging the IP address of the listening server

MS 2.5 Outputs: Successful ping result

MS 2.6 Normal

MS 2.7 Whitebox

MS 2.8 Functional

MS 2.9 Integration

MS 3.1 Agent List Route Test

MS 3.2 This test verifies that the API route for agent list works

MS 3.3 Send an HTTP request to the master server, verify that a list or empty array is sent back

MS 3.4 Inputs: HTTP request with user token

MS 3.5 Outputs: 200 HTTP code with empty list or list of agents

MS 3.6 Normal

MS 3.7 Whitebox

MS 3.8 Functional

MS 3.9 Unit

MS 4.1 Users List Route Test

MS 4.2 This test verifies that the API route for users list works

MS 4.3 Sends an HTTP request to the master server, verify that a list or empty array is sent back

MS 4.4 Inputs: HTTP request with user token

MS 4.5 Outputs: 200 HTTP code with empty list or list of agents

MS 4.6 Normal

MS 4.7 Whitebox

MS 4.8 Functional

MS 4.9 Unit

MS 5.1 HTTP Request Token Existence Test

MS 5.2 This test verifies that all routes on the API require a user token

MS 5.3 Sends an HTTP request to master server without a user token, verify that a Forbidden error message is sent back

MS 5.4 Inputs: HTTP request without user token

MS 5.5 Outputs: 403 HTTP code

MS 5.6 Abnormal

MS 5.7 Whitebox

MS 5.8 Functional

MS 5.9 Unit

MS 6.1 Invalid user token test

MS 6.2 This test verifies that routes check for proper token existence

MS 6.3 Sends an HTTP request to master server without a random field in user token, verify that a Forbidden error message is sent back

MS 6.4 Inputs: HTTP request with invalid user token

MS 6.5 Outputs: 403 HTTP code

MS 6.6 Abnormal

MS 6.7 Whitebox

MS 6.8 Functional

MS 6.9 Unit

AG 1.1 Agent Feedback Test

AG 1.2 This test will verify the functionality of a deployed agent after any updates to the agent software

AG 1.3 Send HTTP request to specified agent and verify successful response

AG 1.4 Inputs: HTTP GET request to agent IP:Port

AG 1.5 Outputs: HTTP GET request returns data from the agent successfully

AG 1.6 Normal

AG 1.7 Blackbox

AG 1.8 Functional AG

1.9 Integration

AG 2.1 Agent HTTP Post Token Existence Test

AG 2.2 This test will verify that all agent specific routes require an agent token.

AG 2.3 Send HTTP request to specified agent without agent token, and verify forbidden response

AG 2.4 Inputs: HTTP POST request to master server IP:Port

AG 2.5 Outputs: 403 HTTP code

AG 2.6 Abnormal

AG 2.7 Whitebox

AG 2.8 Functional

AG 2.9 Unit

AG 3.1 Agent Sanitization Test

AG 3.2 This test will verify that the agent sanitizes metrics parsed from kernel files in order to prevent malicious input from bad actors.

AG 3.3 Modify kernel file contents to contain illegal characters and verify agent sends sanitized payload.

AG 3.4 Inputs: Modified kernel file with illegal/malicious data

AG 3.5 Outputs: Sanitized payload with malicious data removed

AG 3.6 Normal

AG 3.7 Whitebox

AG 3.8 Functional

AG 3.9 Unit

WA 1.1 Web App Login

WA 1.2 This test verifies that the login functionality is working properly

WA 1.3 Login to web app, verify successful login

WA 1.4 Inputs: Test username/password in the username/password fields on the main site

WA 1.5 Outputs: Successful login shown by navigation to the dashboard screen

WA 1.6 Normal

WA 1.7 Blackbox

WA 1.8 Functional

WA 1.9 Integration

WA 2.1 Dashboard CPU Graph Appearance

WA 2.2 This test verifies that the CPU graph is shown on the dashboard

WA 2.3 Login to app, verify CPU graph is shown

WA 2.4 Inputs: Test username/password in the username/password fields on the main site

WA 2.5 Outputs: Visual verification that the CPU graph is shown

WA 2.6 Normal

WA 2.7 Blackbox

WA 2.8 Functional

WA 2.9 Integration

WA 3.1 Dashboard HDD Utilization Graph Appearance

WA 3.2 This test verifies that the HDD Utilization graph is shown on the dashboard

WA 3.3 Login to app, verify HDD Utilization graph is shown

WA 3.4 Inputs: Test username/password in the username/password fields on the main site

WA 3.5 Outputs: Visual verification that the HDD Utilization graph is shown

WA 3.6 Normal

WA 3.7 Blackbox

WA 3.8 Functional

WA 3.9 Integration

WA 4.1 Dashboard RAM Utilization Graph Appearance

WA 4.2 This test verifies that the RAM Utilization graph is shown on the dashboard

WA 4.3 Login to app, verify RAM Utilization graph is shown

WA 4.4 Inputs: Test username/password in the username/password fields on the main site

WA 4.5 Outputs: Visual verification that the RAM Utilization graph is shown

WA 4.6 Normal

WA 4.7 Blackbox

WA 4.8 Functional

WA 4.9 Integration

WA 5.1 Dashboard Network Utilization Graph Appearance

WA 5.2 This test verifies that the Network Utilization graph is shown on the dashboard

WA 5.3 Login to app, verify Network Utilization graph is shown

WA 5.4 Inputs: Test username/password in the username/password fields on the main site

WA 5.5 Outputs: Visual verification that the Network Utilization graph is shown

WA 5.6 Normal

WA 5.7 Blackbox

WA 5.8 Functional

WA 5.9 Integration

WA 6.1 Dashboard Agent List Appearance

WA 6.2 This test verifies the Agent List table is shown on the dashboard

WA 6.3 Login to app, verify agent list table is shown on dashboard

WA 6.4 Inputs: Test username/password in the username/password fields on the main site

WA 6.5 Outputs: Visual verification that the RAM Utilization graph is shown

WA 6.6 Normal

WA 6.7 Blackbox

WA 6.8 Functional

WA 6.9 Integration

WA 7.1 Logout Function

WA 7.2 This test verifies the functionality of the logout function on the dashboard

WA 7.3 Login to app, click logout in upper right corner, verify navigation to login screen

WA 7.4 Inputs: Test username/password in the username/password fields on the main site

WA 7.5 Outputs: Visual verification of navigation to login screen

WA 7.6 Normal

WA 7.7 Blackbox

WA 7.8 Functional

WA 7.9 Integration

PF 1.1 Performance of agents

PF 1.2 This test checks that agents are able to send metrics at the rate of 1 data object per 5 seconds

PF 1.3 Setup agent, check database for proper data collection in a span of time

PF 1.4 Inputs: Agent setup on a server

PF 1.5 Outputs: Proper performance of agent

PF 1.6 Normal

PF 1.7 Whitebox

PF 1.8 Performance

PF 1.9 Integration

PF 2.1 Performance of master server

PF 2.2 This test verifies that the master server correctly handles incoming metric data requests from the various agents

PF 2.3 Setup multiple agents for multiple sources, prepare metrics and send metric data, check database to verify proper load management

PF 2.4 Inputs: Agent metric data

PF 2.5 Outputs: Proper request management of central server

PF 2.6 Normal

PF 2.7 Whitebox

PF 2.8 Performance

PF 2.9 Integration

Overall Test Case Matrix

Test Identifier	Normal/ Abnormal	Blackbox/ Whitebox	Functional/ Performance	Unit/ Integration
MS1	Normal	Whitebox	Functional	Integration
MS2	Normal	Whitebox	Functional	Integration
MS3	Normal	Whitebox	Functional	Unit
MS4	Normal	Whitebox	Functional	Unit
MS5	Abnormal	Whitebox	Functional	Unit
MS6	Abnormal	Whitebox	Functional	Unit
AG1	Normal	Blackbox	Functional	Integration
AG2	Abnormal	Whitebox	Functional	Unit
AG3	Normal	Whitebox	Functional	Unit
WA1	Normal	Blackbox	Functional	Integration
WA2	Normal	Blackbox	Functional	Integration
WA3	Normal	Blackbox	Functional	Integration
WA4	Normal	Blackbox	Functional	Integration
WA5	Normal	Blackbox	Functional	Integration
WA6	Normal	Blackbox	Functional	Integration
WA7	Normal	Blackbox	Functional	Integration

PF1	Normal	Whitebox	Performance	Integration
PF2	Normal	Whitebox	Performance	Integration

Results:

Test Name	Testing Steps Taken	Result
MS1	User logged in on login page, proceeded to agent list view, and clicked create new agent button. User filled in necessary agent preferences and submitted form. User received associated agent files and ran the binary installation exe on their host server. After installation and configuration, the user logged back into their account, navigated to the dashboard view, and selected their registered agent from the agent list sidebar.	PASS – The user was presented on the left sidebar with their new agent’s name and a badge showing the agent was online. The visualizations showed data populating each graphic.
MS2	The master server was configured and activated to receive incoming metric data for each agent. After configuration, a ping was performed to the public IP for the master server.	PASS - The ping was successfully received, and all packets were received with no loss or drops of the sent packets.
MS3	The Jest testing framework was instantiated in the code base to test API endpoints. After installation and configuration of Jest and the Jest testing environment, a jest test was created to get the agents list. The test comprised of sending a GET request to the agent API to get the agents list.	PASS – The test was successful as the status code returned was a 200-status indicating success and the body of the message data was greater than 0. The body returned was a list of agents.

MS4	See MS3 Agent 3 for test as this test was not necessary for the true list API test results since there is no user lists functionality in Heimdall.	N/A – The MS3 test covered the only agent/user list functionality included in the final version of the Heimdall application.
MS5	Multiple jest tests were created to test every API endpoint that required user authentication and making sure the user secret token was not sent in the various HTTP POST requests. This included tests for agent download, getting the agent list, etc.	PASS – The Jest test produced a passing result. Each API request, without the user secret token, produced a general 500-status code indicating failure and the error returned each time was specifically a 403-error code as expected.
MS6	Jest tests for all user authentication required API endpoints already created were modified to send the various HTTP requests with the user secret token, except that one required field was left out for each test. The field left out was attempting to be randomly removed for each test.	PASS – The jest test produced a passing result. The tests produced the 500-status indicating failure and, in every case, the specific error that caused the internal server error was a 403-status code due to the missing field as was expected.
AG1	The user created an agent and installed and configured it on their host server. Once the agent was deemed to be online as seen by the user on the dashboard, a jest test was run to send an HTTP Get request to the agent's IP address and to configured port for the agent.	PASS – The Jest test produced a passing result. The agent received the HTTP request and successfully sent back data and produced a 200-status code indicating success. The resulting body of the data returned for the GET request was the expected agent data structure with the collected metric fields.
AG2	Jest tests were created for all API endpoints that dealt with using POST requests with agent data metrics to the master server and required an agent token for authentication and making sure the agent token was not	PASS – The POST requests to the master server public IP and configured port produced a 500-status code in every case indicating failure. The specific code, in every case, was a 403-status code due to

	sent in the various requests. This included the base metric data structure, the host information, network metrics, etc.	the missing agent token in each request as expected.
AG3	The user, in a test environment that was hosting their agent, modified their kernel dealing with host system information to reflect structured data except with various “junk” malicious information that was not of use to the agent and its structured host metrics.	PASS – The body of the agent data passed to the master server only included the developer defined metrics in the host information structure and did not include the newly added kernel information by the user as expected.
WA1	The user, having just created a new account for Heimdall, was navigated back to the login page. The user then entered their defined account credentials into the respective username and password fields.	PASS – After entering credentials and clicking the login button, the user was successfully navigated to the dashboard view interface for Heimdall’s web application.
WA2	The user, having a pre-existing Heimdall account, logged in to the Heimdall web application via the login page. After logging in, the user was redirected to the dashboard view interface. The user selected one of their registered agents.	PASS – The CPU utilization graph was clearly and correctly presented at the top of the screen for the user to view their selected agent’s collected CPU utilization metrics over different time frames as expected.
WA3	The user, having a pre-existing Heimdall account, logged in to the Heimdall web application via the login page. After logging in, the user was redirected to the dashboard view interface. The user selected one of their registered agents. The user scrolled down to the bottom of the page.	PASS – The HDD Disk Information graphic was presented and clearly and correctly displayed the hosting server metrics for each partition such as path, used space, and free space as expected.
WA4	The user, having a pre-existing Heimdall account, logged in to the Heimdall web	PASS – The RAM utilization graph was present in its spot below the CPU utilization

	application via the login page. After logging in, the user was redirected to the dashboard view interface. The user selected one of their registered agents.	graph and the RAM graph clearly and correctly presented the memory utilization metrics collected from the selected agent for the user to see as expected.
WA5	The user, having a pre-existing Heimdall account, logged in to the Heimdall web application via the login page. After logging in, the user was redirected to the dashboard view interface. The user selected one of their registered agents.	PASS – The Network utilization graph was present below the CPU utilization graph and the Network graph clearly and correctly presented the network utilization metrics collected by the current agent for the user to see as expected. The user was presented with a dropdown to switch between the host system's network adaptors and the graph correctly updated to present accurate metrics for each adaptor for the user to see as expected.
WA6	The user, having a pre-existing Heimdall account, logged in to the Heimdall web application via the login page. After logging in, the user was redirected to the dashboard view interface.	PASS – On the left side of the screen, the sidebar reflected the agent list of every agent registered by the user along with badges indicating their online/offline status as expected.
WA7	The user, having a pre-existing Heimdall account, logged in to the Heimdall web application via the login page. After logging in, the user was redirected to the dashboard view interface. The user then clicked the top right logout icon. The user clicked on the toolbar presenting a "logout" prompt.	PASS – The user, after clicking the "logout" prompt, was successfully logged out of the web application and redirected to the login page as expected.
PF1	The user, having a pre-existing account, logged in to the Heimdall web application. The user then navigated to the agent list page and	PASS – The agent collected data was being successfully sent at the user-defined polling rate at the time of creation. The data object

	<p>clicked the create a new agent button. The user then installed the agent on their target server and configured the agent for metric collection. After installation and configuration, the master server was inspected.</p>	<p>was successfully sent every 5 seconds; however, the user defined a polling rate of 1000 ms and the data object was also being sent successfully at this rate as well. The database was successfully populating each table that required insertion of updating of metric records for the current agent.</p>
PF2	<p>The user, having an existing account, logged in to the Heimdall web application via the login page. The user then navigated to the agent list page and created multiple agents. The user then installed the agents on multiple target servers and set the agents' configurations. After collection for about a half hour, the user changed the host of one of the agents.</p>	<p>PASS – The collected metrics from the several agents were handled correctly by the master server. The associated records, ids, and various metrics were all accurately inserted for the agents and upon further collection were correctly updated in the database. For the changed agent, the host information in the master server was correctly upserted, meaning that since the agent id already existed rather than inserting new records the records corresponding to the agent were updated instead, for host information, node information, network adaptor information, etc. according to the host system change as expected</p>