Automatic Bug Logger Design Document

Version History:

S. No	Description	Author	Version	Date Modified
1	Automatic Bug Logging Design Document	Santhosh	0.1	5th March 2014

Objective : An Automated Bug Logging Mechanism for Automation runs.

Problem Statement: Currently we dont have an automated way of logging bugs under bug managment system say jira post a run. With this facility, it will be possible to log bugs post the run automatically.

Few Constraints:

1. Automation runs happens across Geos, different nodes run automation for a given build for various hypervisors either same or different on these nodes.
2. For the same build, different categories of tests happens for different configurations some times differently.
3. Each node running automation, post its run if automatically logs bugs to jira, then multiple nodes are logging bugs to jira at a given time is possible. These runs could be of same type some times different, bugs for same build, same hypervisor etc are possible.
4. If there are re runs, and parallel runs across geos and different systems, then we are duplicating the bug logging effort if logged from these nodes.
5. Bug look up on jira is equally costly for every run, throttling the lookup operations and amount of data retrievals for number of API calls is equally costly. Jira limitation to do number of lookups is throttled.
6. Network connectivity issues to jira from node Is possible at the time of bug logging. Jira server going down at the time of bug logging etc are possible scenarios.
7. If a bug logging fails at a given time, then reattempt from the node is required, but this approach makes the node not to use for further run till bugs are logged or to say node will be blocked.
8. Avoiding multiple lookups and logging bugs from multiple nodes simultaneously for runs needs to be avoided.
9. Issues reported from automation run can be either be test failure or a product failure. So, we may want to log failures for test as well as bugs. This will help fixes for those issues equally possible for automation runs.
10. Log uploads for a given bug from multiple nodes, is unnecessarily increasing the network traffic for same build run.
11. Currently, as part of bug report, adding test case steps to reproduce, is not possible as we dont have that information as part of any TMS(Test Management Solution).

Design Flow for Logging Bug Automatically:

1. We will use a vm machine(from here onwards called Bug Logger Machine), configured with preinstalled scripts to log bugs on behalf of multiple nodes.
2. We will use nosetest automation run xunit output to collect test run information for logging bugs automatically.
3. We will use nfs share to have all logs uploaded and use this information from BugLogger Machine to attach logs to bugs.
4. We will use the jenkins infrastructure configured with jobs as part of workflow for a given

automation run to log bugs.

5. The Bug Logger Machine maintains a local database to store few key information for bug logged or yet to be logged and its state along with build information.
6. The database is a light weight one, and will be used to store build information along with bugs pertaining to that build.
7. For every given run, we will categorize the failures in to two sets, one reported as test errors and other reported as product failures.
8. We will use the nfs share uploaded with all logs collected from a given run. The nfs share will be mounted on Bug Logger Machine to extract logs to be attached to bug report. All logs including management server logs, tc logs and other information are uploaded to this share post automation run using jenkins infrastructure.
9. The Bug Logger Machine will have few commands exposed as web service.
10. Each node under jenkins configured to run automation run, will call the webservice with version,build and hypervisor type to trigger the bug logging on Bug Logger Machine.
11. Each node as part of automation run has a job configured under jenkins, to upload the logs for that run to the nfs share.
12. A new job under jenkins will be added to trigger the webservice on Bug Logger Machine. The job runs post the log upload job(mentioned under 11) as downstream project .
13. The node calls the webservice command on Bug Logger Machine to initiate bug logging job for that build.
14. The Bug Logger Machine has a queue maintained to store the job information once the request is received.
15. The Bug Logger has an event handler for this queue, and works post an entry added to trigger the bug logging.
16. The Bug Logger job is triggered as part of its flow with an entry from this queue, extracts information from nfs share based upon the arguments provided as part of service request i.e., **<version, hypervisortype,build_no,bvt>**
17. The Bug Logger job triggered, then creates a temporary work space for that jobid.
18. Under this workspace, it extracts the information from nfs share and collects the test run information.
19. The job extracts various information and segregates the results in to two buckets as mentioned viz., errors and failures.
20. The results under these buckets are initially searched inside the local DB for their state EX: open or reopened or resolved etc.
21. If a bug is alrleady logged for that test case in the local db and is not resolved yet for that version, then it will not be logged as a bug.
22. The key for each test case is **<csversion.hypervisortype.component.testmodule.testclass.testname>**
23. If a bug is not logged or not available under the local db, then it is added to bug queue to be logged as new bug.
24. If a bug is already in open\reopen state, a comment to bug will be added stating the bug is reproducible in this build as well.
25. Logging test errors under jira for test code fixes happens similarly as mentioned above. Logging test errors is configurable and bugs for test failures are logged only when configured to be logged.
26. Each bug report will have a subject mentioning its key and failure information collected from test results trace file as part of its description.
27. Each bug report will have logs attached as part of it when logged.
28. A jira handle based upon the configuration is first created as part of the bug logger job post bug

buckets are created.

29. If jira connection once checked is not available, the bug logger job will then store these information to be logged later in to db.
30. If jira connection is available, then bug logger job will do a look up on jira for their existence before logging and log bugs accordingly.
31. All successfully logged bugs are then stored back in to DB with their state, for future jobs to avoid jira lookups.
32. A back ground job configurable with specific time interval, fetches all open bugs inside the local db and updates their state with that of available under jira
33. The above step is useful to update the state so that bug logger job can get latest state of that bug everytime it does a local lookup before logging.
34. Each bug logger job as well has local run log maintained at a configurable path. So, users can see for the state of bug logging post run.
35. The created work space will be delted post the successful bug logging is done.
36. An Email will be sent to configured email address with count of bugs logged along with other metadata post the job finish.
37. The bug logger job status inside the db will be updated post it is finished.

Configurations:

```
'''
Config Part of Bug Filing
'''

BugLoggerConfig = {
   "BUG_TRACKER_SRVR_DETAILS":
   {
      "URL": "https://issues.apache.org/jira",
      "USER": "santhoshe",
      "PASSWORD": "password"
   },
   "FAILURE":
   {
      "REPORTER": "santhosh.edukulla@citrix.com",
      "ASSIGNEE": "santhosh.edukulla@citrix.com",
      "PROJECT": "cloudstack",
      "COMPONENTS": "Automation"
   },
   "ERROR":
   {
      "REPORTER": "santhosh.edukulla@citrix.com",
      "ASSIGNEE": "santhosh.edukulla@citrix.com",
      "PROJECT": "cloudstack",
      "COMPONENTS": "Automation"
   },
   "MISC_DETAILS":
   {
      "ENVIRONMENT": "NA",
      "DESCRIPTION": "Test Case Failed. Attached are CS and Test Run Logs",
```

```
      "AFFECTED_VERSIONS": "4.3",
      "PRIORITY": 3,
      "STATUS": "open",
      "RESOLUTION": "open",
      "COMPONENTS": "Automation",
      "CS_LOG_PATH": "/tmp/CSLogs/",
      "TEST_LOGS": "/tmp/MarvinLogs/",
      "TYPE": "Bug",
      "COMPRESSED": "yes",
      "TEST_OUT_FOLDER_MNT_PATH":"/home/nfs_mnt_bug_path/",
      "BUGS_OUTPUT_PATH":"/home/",
      "BUG_EXCLUSIONS":"/home/bug_exclusions/",
      "CS_LOG_FILE_NAME":"management-server-logs.gzip",
      "TC_RUN_LOG_FILE":"tc_run_log.gzip",
      "EMAIL_USERS":"santhosh.edukulla@citrix.com",
      "FROM_EMAIL_ADDRESS":"santhosh.edukulla@citrix.com",
      "LOG_ERROR_BUGS":"yes",
      "LOG_PATH":"/home/"
   },
   "DB_DETAILS":
   {
     "DB_HOST":"localhost",
     "DB_USER":"root",
     "DB_PASSWD":"password",
     "CATALOG":"cloud_bugs"
   }
}
```

All the configurations above are self explainatory.

WebService Calls:
Following are few webservice calls exposed on Bug Logger Machine. All are http calls currently.

Command	Request Args	Response Args	Description	Example	
LogBugs	[version, build, hyptype, type]	[Jobid, SUCCESS\FAILED]	Called from Jenkins Automatically post the log upload job. A triggering job to start bug logging	Http://<Bug Logger Machine>/MarvinBugLogger/cmd=LogBugs?version=4.3&build=build-1234&hyptype=xen&type=bvt	
GetJobStatus	Jobid	SUCCESS\FAILED	Can be used by users to know the particular job status	Http://<Bug Logger Machine>/MarvinBugLogger/	

				cmd=GetJobStatus?jobid=1	
UpdateBugs	List<bugs_to_be_updated>	SUCCESS\FAILED	Can be used by users if they want to update the given bugid with status on BugLoggerMachine, when they manually change the bug status on JIRA	Http://<Bug Logger Machine>/MarvinBugLogger/cmd=UpdateBugs?bugid=1&bugid=2	

BackGround UpdateJob:
This job is responsible for updating the status of all bugs under Bug Logger Machine DB from jira. Configured to run at specific intervals and runs on Bug Logger Machine.

DB Schema(Sample):

```
CREATE TABLE `cloud_bugs`.`bugs`(
  `id` bigint unsigned NOT NULL UNIQUE AUTO_INCREMENT COMMENT 'id',
  `bugid` varchar(40) NOT NULL UNIQUE,
  `testname` varchar(1024) NOT NULL,
  `component` varchar(24) NOT NULL,
  `version` varchar(10) NOT NULL,
  `result`  varchar(15) NOT NULL,
  `message` varchar(1024) NOT NULL,
  `type` varchar(10) NOT NULL DEFAULT "bvt",
  `hwtype` varchar(10) NOT NULL,
  `subject` varchar(1024) NOT NULL,
  `trace`   TEXT CHARACTER SET binary,
  `created_date` TIMESTAMP NOT NULL  DEFAULT CURRENT_TIMESTAMP,
  `updated_date` TIMESTAMP NOT NULL  DEFAULT 0,
  `state` varchar(24) NOT NULL,
  `buildno` unsigned NOT NULL Default 0,
  PRIMARY KEY (`id`),
  CONSTRAINT `bugs_bugid` UNIQUE (`bugid`),
  CONSTRAINT (`bug_build_key`)  FOREIGN_KEY buildno references `build`.`jobid`
  INDEX `cloud_bugs_testname`(`testname`) ,

) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `cloud_bugs`.`build` (
  `jobid` bigint unsigned NOT NULL UNIQUE AUTO_INCREMENT COMMENT 'id',
  `version` varchar(10) NOT NULL,
  `buildno` varchar(24) NOT NULL UNIQUE,
  `type` varchar(24) NOT NULL,
  `hyptype` varchar(24) NOT NULL,
```

```
  `file_paths` varchar(1024) NOT NULL default '',
  `state` varchar(12) NOT NULL default "started",
  `created_date` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updated_date` TIMESTAMP NOT NULL DEFAULT ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

New Packages Used away from default:

All the below packages are installed on Bug Logger Machine.

S.No	Name	Description
1	Jira-Python	Used for connecting to Jira
2	Xunit Parser	Used for parsing the xunit data
3	Sqlalchemy	Used for Data Mapper pattern to interact with DB
4	Pycrypto,http lib ,pytls etc	For exposing few api calls and for oauth authentication

New Modules and Interfaces Created:

1. MarvinBugLogger: Main Class for handling bug logger job and logic for comparing,retrieving final bug list to log bugs etc
2. MarvinBugLoggerCmdManager: Handles Web API calls.
3. MarvinBugLoggerDal : Handles SQL operations
4. MarvinRemoteBugManager : Jira interface to client
5. MiscHandler: Handlesfile attachments, compressing result,trace data in between interacting with jira,handling notifications.

NFS Share Changes:

Below are few changes required under jenkins to upload the test run, logs to nfs for a given release version,hypervisor and build.

1. We need to have logs and other information uploaded to nfs share for every run as per below format.
   nfsip://path/version/hypervisorType/build
   EX: nfs://10.0.0.1:/4.3.0.0/vmware/Build-12304.
2. Added to note1, Under the path (EX: 4.3.0.0/vmware/Build-12304) for every run on nfs share, we will upload the following logs to it as part of every run (bvt or regression) with the below formats

   Management server logs zipped as file name "management-server-logs.gzip",
   Test Results (xunit for complete run) as "test_results.gzip"
   Test Case Run log(MarvinLogs) as "tc_run_log.gzip"

Misc Items:

- The Required Scripts on Bug Logger Machine will be monitored by a monitor job to keep the availability of scripts wherever possible.
- DB HA through Backup will be done periodically.
- Availability for Bug Logger Machine will be implemented in future.

Flow Diagram: NA(will add later)