1. Summary of Changes
2. Requirements
3. Functional Requirements Specification
4. Effort Estimation
5. Domain Analysis
6. Interaction Diagrams
7. System Architecture and System Design
8. Reference
9. Summary of Changes

Project Changes

* Dependencies changed to include apache ignite and spring and 4j logger and 4j2 logger.
* Idea of a single server per chain has been replaced with a new model of a cluster of servers each with the information shared between them.
* Including the idea of deploying tasks and jobs from a database rather than coding them into Ignite.

Documentation Changes

* Requirements have been update, assumptions have been updated the model of the architecture has been created.
* There is now a UI update portion of documentation specifically geared towards addressing the changes in layout and information representation from checks.
* User interaction diagrams have been updated and created.

1. System Requirements

Enumerated Functional Requirements

Weights (higher = greater priority)

3 = Primary feature – Required for success

2 = Required for proper function of primary features

1 = Secondary feature – for convenience

|  |  |  |
| --- | --- | --- |
| Identifier | Priority | Description |
| REQ-1 | 3 | Bootstrap MkvIgnite – Bridge data from Marketview to Ignite |
| REQ-2 | 3 | Understand how to write jobs in the Ignite cluster |
| REQ-3 | 3 | Understand the requirements that support has for improvements to the Glance UI |
| REQ-4 | 2 | Have a cluster of nodes setup to partition cache data across |
| REQ-5 | 1 | Make a thin client that does not change or influence the topological structure of ignite for tasks. |
| REQ-6 | 2 | Create runnable jobs within Ignite that work on cached data |
| REQ-7 | 1 | Make a thin client that does not change or influence the topological structure of ignite for tasks. |
| REQ-8 | 2 | Retrieve data in an SQL inquiry to view cache and manually see data |
| REQ-9 | 3 | Manipulate data within the cache for checks from anywhere in the cluster |
| REQ-10 | 2 | Make the clients (thin or not) read only to the data. Exclusive write access only for the server that maintains the Ignite Cache for a specific chain |
| REQ-11 | 1 | Make search functionality so that users can search for their desired chain and/or record. |
| REQ-12 | 2 | The system will schedule jobs or tasks to be completed in a user desired fashion |
| REQ-13 | 2 | Load jobs from a separate database so that integrating with Ignite is not cumbersome |
| REQ-14 | 2 | The System will allow users to print the cache |
| REQ-15 | 3 | The system should update the cache when new information is available |
| REQ-16 | 1 | Data and Metric logging for past analysis |
| REQ-17 | 3 | Horizontal scalability (ability to seamlessly add hardware and server to accompany growth of data and service. |
| REQ-18 | 1 | The system should manage the load across the cluster and prioritize certain jobs |
| REQ-19 | 2 | The system will subscribe to all market view data available and distribute it among other servers in the cluster |
| REQ-19 | 3 | The system will be able to recover from random node disconnections and crashes within the cluster |

Enumerated Nonfunctional Requirements

F.U.R.P.S. =

**Functionality**

The F in the FURPS+ acronym represents the main product features that are familiar within the business domain of the solution being developed. The functional requirements can also be very technically oriented. Functional requirements that you may consider to be also architecturally significant system-wide functional requirements may include auditing, licensing, localization, mail, online help, printing, reporting, security, system management, or workflow.

For our internal tool, the main features would be:

* A service to run market view checks on
* A service that caches Marketview data to make it more easily and readily available
* Cluster computing limiting client computation and client cache access
* Easier and abstracted cache access
* Easy interactive UI that helps the user understand the data presented to them
* Fetch specific chains and records asked by the user
* Data logging and metric logging for performance management

**Usability**

Usability includes looking at, capturing, and stating requirements based around user interface issues — things such as accessibility, interface aesthetics, and consistency within the user interface.

User Interface Accessibility – To facilitate user access, glance 2.0 will validate user information via SSO account. This avoids the need to create a new account and should be easier for account management for both the user and system managers to alter accessing rights.

Distinctive Features – To increase the productivity the user, Glance 2.0 will display all suite nodes, filtered by the user’s choice, in a table organized by the root categories. Each node tile displayed will allow the user to view the most immediate cause of failure by hovering over it and the full list of all entries included within the node by clicking on each node.

User Interface Consistency – Continuing the design of Glance, Glance 2.0 would maintain a simplified layout by laying out only the essentials to the user on the immediate home page. Any new data fields added should be consistent with the existing tables seen within each node page. All new features are only adding new outlets to redirect users to immediate failing nodes in a more efficient manner without altering the current template.

**Reliability**

Reliability includes aspects such as availability, accuracy, and recoverability — for example, computations, or recoverability of the system from shutdown failure.

Availability – As our system will be run off of the companies computers, availability will be only when the computer is on and functioning. When a system terminates it can be restarted with the cached backed up across multiple different nodes. This should allow for high availability among all nodes and data; however, speed may suffer in such instances.

Accuracy – The accuracy of the Ignite data will be at the mercy of three factors, the network connections with MKV, node cache speed, and MKV itself. If there are slow network conditions, nodes may not update in a timely manner. If there is a problem with MKV obviously Ignite cannot fix the incorrect data. Finally, if the nodes or systems are overburdened they may lag out and disconnect meaning when they restore the data may be old or incomplete.

Recoverability – The program is mainly server based. As long a single server exists node can connect and reconnect/sync to MKV and continue operating. If a node (server side) crashes, then when it reconnects it will automatically sync state with the cluster and resume. The cluster will hide this crash from the user maintain operation with failover precautions for tasks and data. Finally, in the case bad code is pushed servers can be timed out to automatically restart to recover from indefinite loops or long query times.

**Performance**

Performance involves things such as throughput of information through the system, system response time (which also relates to usability), recovery time, and start-up time.

Speed – The servers have a lot of initial handshakes to preform upon entering the cluster meaning that server nodes may take a few seconds to boot up, connect and sync with Marketview and other nodes. However, client nodes should be relatively quick with startup considering they do not create caches or clusters. The only metric is the initial handshake which is about 24 Bytes of data. Upon each new client node entering the grid, the topology will be updated resulting in a few milliseconds to sync that information; however, there is not topology changes with thin clients resulting in an even faster start up.

Efficiency – Ignite will pool available threads and distribute tasks in a multi-threaded fashion keeping track of distributed deadlocks and live locks; with optimization load sharing and throttling will be available in order to increase throughput.

Resource Consumption – The system uses a lot of RAM; however, this can be scaled and configured. The amount to RAM and disk space is dependent on how much information from MKV ignite we receive. As for CPU computation, that is dependent on the task run within the cluster, the cluster will fully utilize the CPU including all available threads.

Scalability – The System is incredibly scalable. Since the servers in the cluster act as an abstracted object the topology will not change externally, so servers can be added to accommodate for additional resources. Since loads may vary depending on checks and times this system can add or remove nodes based on performance requirements making it elastic.

**Serviceability**

Repair Speed – We are running on Bit Bucket so in the unfortunate case that an error does arise then we can always revert back to a state where the client/server worked. This allows us time to repair while decreasing the time in which the service is down. The speed of which the repair occurs also increases with the use of Bit Bucket since we can see exactly what lines were changed between the previous version and the damaged one. This allows for immediate analysis and repair.

Install ability – Since the users are accessing the information via web servers the installation should be relatively simple. For deployment on the server side, deploying the RPM’s on machines will handle the instillation and connection between cluster nodes

Testability – Since data is available through Marketview we can test how the servers handle large amounts of data for processing we can also test by simulating heavy client usage. For topology and update speeds we can create multiple servers and clients and have them join/leave the system thus forcing topology adjustments. As for regular runtime operation we have logs and metrics allowing us to see where and at what time errors occur

Configurability – Our code is configurable since it relies on the apache ignite cluster model where node are continually dropping and joining. We can add or remove clients, and servers without an interrupt in service, as well as handle network partitioning of servers, so machines can run remotely. Additionally, the ability to deploy code to the system allows us to configure what tests are running and schedule task making the system configurable to a multitude of scenarios.

**On-Screen Appearance Requirements**

The main point of this project is to minimize the need of navigation between the numerous internal nodes to find the fault suite node / entries. We have changed how each node is accessed while maintaining the previous structure to allow users to scrutinize the hierarchy between each node.

Home Page



Show Closed

Show Undefined

Show Pass

Show Warn

Show Fail

Name, Description, Comments

Name, Description, Comments

Name, Description, Comments

Name, Description, Comments

BondPricer  
Known

BondPricer  
Critical

BondPricer

IMM

This element consists of two main parts: the existing part of three (or more) root nodes that demonstrates the hierarchy between all internal and suite nodes and the table below that shows immediately all suite nodes that satisfies the criteria set out by the user in the check box immediately above the row for each designated category.

For all nodes that show in the category row, all would have the following feature: when the user hovers the mouse over each node, a small message box should pop up and display the top two entry information, including entry name, description, and comments.

Any nodes that filter as FAIL would have the following additional features: any node that is marked as critical would have a black shadow box around the node and be placed on the top of the list to draw immediate attention from the user. Any node that is marked as known issue would have the message box change into a different color until the user disables the feature.

Dashboard

**Last Modified**

**Root**

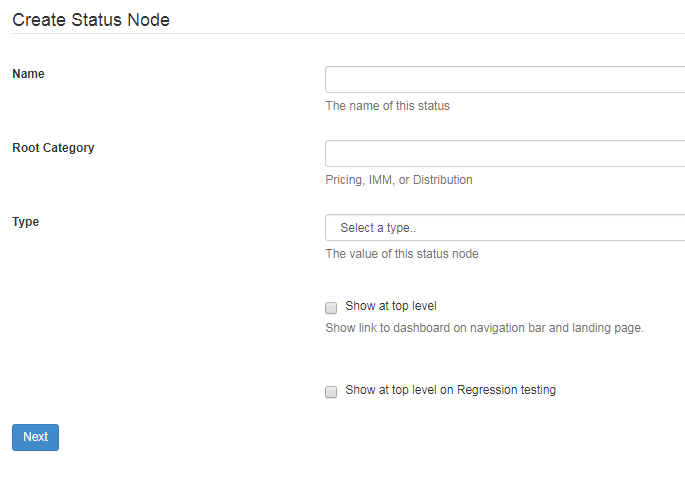
**Show on Regression Page**

**Show on Home Page**

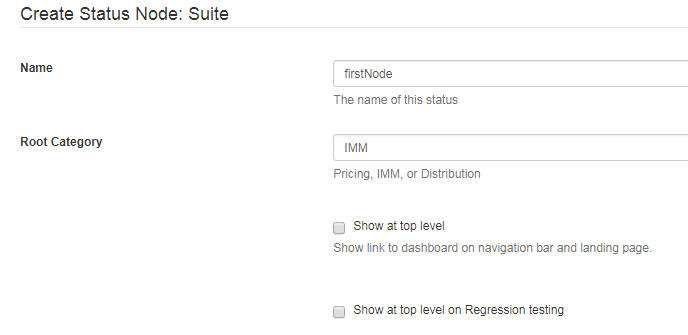
**URI**

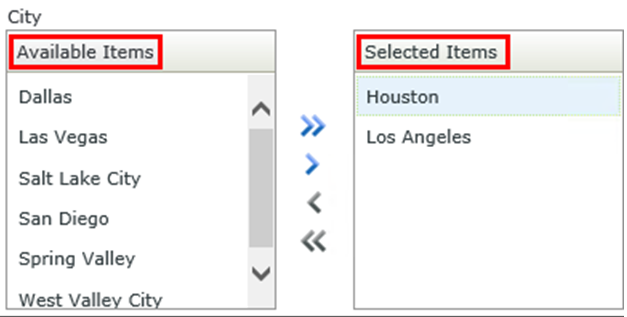
**Name**

The immediate table of node components should show all the relevant information of each node, regardless of the type. The new feature added is to show the root category that each node is part of.

**Add Node**  
  


When adding a new node, this form requires the user to input the name and root category, which are both case sensitive, that would later be needed to identify the node. The two types of node are internal and suite nodes. An internal node is a wrapper and can be thought of as a folder for other internal or suite nodes. A suite node is the main component that holds all of the entry level data.





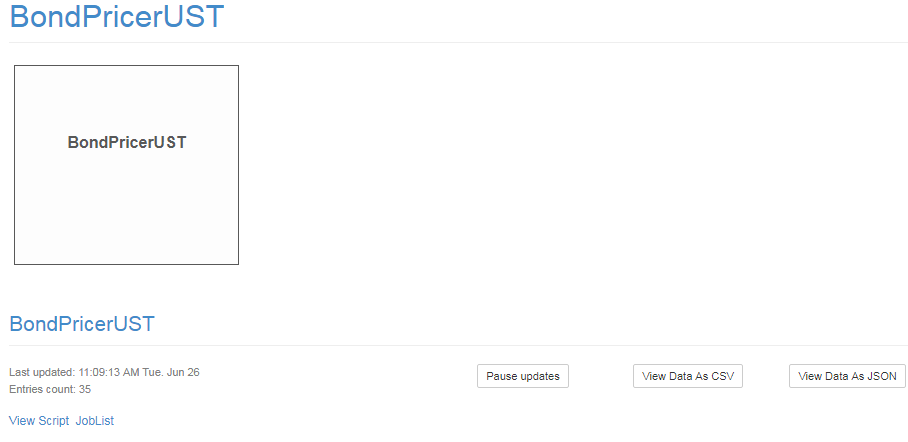
fifthJob

firstJob  
secondJob  
thirdJob  
fourthJob

Suite

The next image allows the user to select specific suite or internal nodes that will belong to the newly created node. For each internal node, the limit for the number of suite nodes is 50, and for each suite node it can only consist of one job check. A new feature added to this page is that we altered the old multiple selection box to this new one to minimize the difficulty of editing and adding multiple nodes into an internal node when necessary.

**Suite Node**



Started

Last Check Reported

Comments

Description

Status

Name

Edit text

3MO

Closed

-  
-

-

-  
-

18:00:00 PM  
Tue, Jan 26 2018  
(UTC Tue 03:16:14)

Known Issue

This is where the user sees all individual status of each entry checks defined in the script. The different possible status types are: Pass, Warn, Fail, Closed, and Undefined. Inside the description it usually shows any relevant information received as part of the check to help with any necessary debugging. Comments is a new feature that should allow the users to edit any text messages that would show when the user hovers over the suite node and a check box below to mark a particular entry as a known issue.

Check Jobs

Under the drop down box in Glance 1.0, there should be three options: Jobs List, Add new Job, and Groovy Script Generator. However, since we are moving check jobs and have them done in Apache Ignite, it would be made consistent with the Dashboard page where no dropdown box is necessary and shows directly the page of check jobs list.

**Check Jobs List**

**Edit**

**Last Modified by**

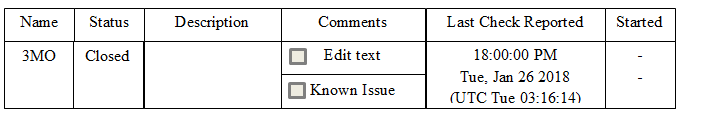
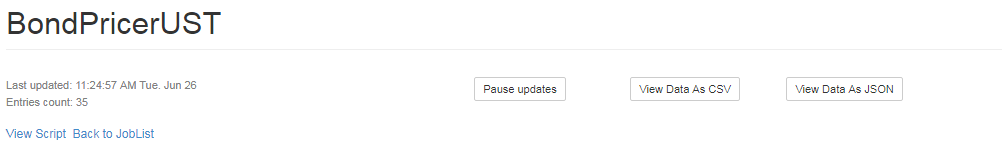
**Delay**

**Version**

**CSV Result**

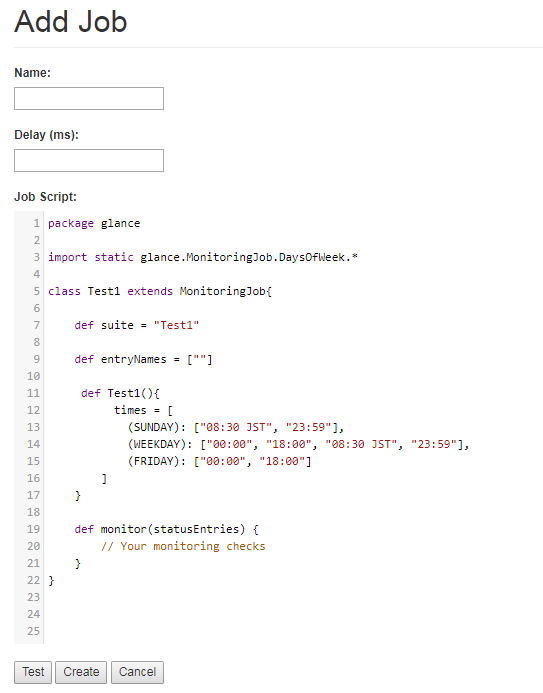
**Name**

**Check Job**



Clicking on the name of any job checks should link the user to this page. It is the same exact template as it would be for a suite node, with the exception of a pure text for name indication on the top left.

**Add / Edit Job**

  
This is the current template provided to production support for implementing new checks. However, as we are shifting all check jobs to be done in Ignite, there is a high possibility that this will no longer be needed.

**Graphite Components**

Graphite is the main pricing application. Under the JMX Connection List, it lists all of the checks that connect to an instance of Graphite, which the data received will be used to feed the charts and tables within Glance.

**Regression Testing**

            Once the user enables a particular node to show on the regression testing, then it would show on this page. The main difference between a regular node in home and in regression testing is that within each suite node, the checks return data from both PROD and UAT to allow the user to compare data before and after implementing a new feature.

1. Functional Requirements Specification

**Stakeholder**

* Experienced
* Professionals
* Production Support
* Development Teams
* Analysts

|  |  |  |
| --- | --- | --- |
| Identifier | Priority | Description |
| REQ-1 | 3 | Bootstrap MkvIgnite – Bridge data from Marketview to Ignite |
| REQ-2 | 3 | Understand how to write jobs in the Ignite cluster |
| REQ-3 | 3 | Understand the requirements that support has for improvements to the Glance UI |
| REQ-4 | 2 | Have a cluster of nodes setup to partition cache data |
| REQ-5 | 1 | Make a thin client that does not change or influence the topological structure of ignite for tasks. |
| REQ-6 | 2 | Create runnable jobs within Ignite that work on cached data |
| REQ-7 | 1 | Make a thin client that does not change or influence the topological structure of ignite for tasks. |
| REQ-8 | 2 | Retrieve data in an SQL inquiry to view cache and manually see data |
| REQ-9 | 3 | Manipulate data within the cache for checks from anywhere in the cluster |
| REQ-10 | 2 | Make the clients (thin or not) read only to the data. Exclusive write access only for the server that maintains the Ignite Cache for a specific chain |
| REQ-11 | 1 | Make search functionality so that users can search for their desired chain and/or record. |
| REQ-12 | 2 | The system will schedule jobs or tasks to be completed in a user desired fashion |
| REQ-13 | 2 | Load jobs from a separate database so that integrating with Ignite is not cumbersome |
| REQ-14 | 2 | The System will allow users to print the cache |
| REQ-15 | 3 | The system should update the cache when new information is available |
| REQ-16 | 1 | Data and Metric logging for past analysis |
| REQ-17 | 3 | Horizontal scalability (ability to seamlessly add hardware and server to accompany growth of data and service. |
| REQ-18 | 1 | The system should manage the load across the cluster and prioritize certain jobs |
| REQ-19 | 2 | The system will subscribe to all market view data available and distribute it among other servers in the cluster |
| REQ-20 | 3 | The system will be able to recover from random node disconnections and crashes within the cluster |

**Actors and Goals**

|  |  |  |
| --- | --- | --- |
| Actor | Actor’s Goal | Use Case |
| User | To access chain, record and information in the ignite server. See the status of checks from the web server | REQ-1, REQ-3, REQ-6, REQ-8, REQ-11, REQ-14, REQ-19 |
| Ignite Server | Connect and manage other nodes, run jobs (checks), manage the cache, and handle request from the user | REQ-4, REQ-9, REQ-12, REQ-15, REQ-17, REQ-19, REQ-20 |
| Web Sever | Push checks into ignite thin client information to Glance | REQ-3, REQ-12, REQ-13 |
| Ignite Thin Client | Receive jobs to run, run the jobs on the server and get the results ,return information to the web server | REQ-2, REQ-5, REQ-6, REQ-7, REQ-12 |
| Ignite Client | Make SQL requests to the server and allow the user to access and monitor the cluster | REQ-8, REQ-11, REQ-14 |
| Market View | Provide data to the ignite cluster about chains and records | REQ-1, REQ-19 |

**Use Cases**

Casual Description

Use Case : Casual Text Description :: Corresponding requirements

UC-1 : A user may want ignite to start subscribing to a specific chain :: REQ-1, REQ-4, REQ-8, REQ-11, REQ-14, REQ-15, REQ-19

UC-2 : A user may want to run specific code from another source on the ignite cluster to check the data within ignite :: REQ-1,REQ-2, REQ-5,REQ-6, REQ-13

UC-3 : A user may want to see the status of a job/task in ignite :: REQ-1,REQ-2, REQ-5,REQ-6, REQ-13, REQ-16

UC-4 : A user may want to connect a client and run code in the cluster at different priority levels :: REQ-1, REQ-2, REQ-5,REQ-6, REQ-12, REQ-13, REQ-18,

UC-5 : A user may want to easily access a record, chain, job or a status :: REQ-1, REQ-3,REQ-8, REQ-9, REQ-10,REQ-11, REQ-15, REQ-19,

UC-6 : A user may want to access server information at any time :: REQ-1,REQ-15, REQ-18,REQ-20

UC-7 : A user may want to access server metrics for health or efficiency analysis :: REQ-1, REQ-5, REQ-16, REQ-20

UC-8 : A user may want to add or delete server nodes to adjust computation performance and cache space :: REQ-4, REQ-15, REQ-16,REQ-17, REQ-20

**Use Case Diagram**

<<include>>

<<include>>

<<include>>

<<extend>>

Market View

<<include>>

<<include>>

Database

<<include>>

<<extend>>

User

<<include>>

<<include>>

<<extend>>

Ignite Server

Ignite Client

**Traceability Matrix**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **REQ’T** | **Pts** | **UC-1** | **UC-2** | **UC-3** | **UC-4** | **UC-5** | **UC-6** | **UC-7** | **UC-8** |
| **REQ-1** | **3** | X | X | X |  | X | X | X |  |
| **REQ-2** | **3** |  | X | X | X |  |  |  |  |
| **REQ-3** | **3** |  |  |  |  | X |  |  |  |
| **REQ-4** | **2** | X |  |  |  |  |  |  | X |
| **REQ-5** | **1** |  | X | X | X |  |  | X |  |
| **REQ-6** | **2** |  | X | X | X |  |  |  |  |
| **REQ-7** | **1** |  |  |  |  |  |  |  |  |
| **REQ-8** | **2** | X |  |  |  | X |  |  |  |
| **REQ-9** | **3** |  |  |  |  | X |  |  |  |
| **REQ-10** | **2** |  |  |  |  | X |  |  |  |
| **REQ-11** | **1** | X |  |  |  | X |  |  |  |
| **REQ-12** | **2** |  |  |  | X |  |  |  |  |
| **REQ-13** | **2** |  | X | X | X |  |  |  |  |
| **REQ-14** | **2** | X |  |  |  |  |  |  |  |
| **REQ-15** | **3** | X |  |  |  | X | X |  | X |
| **REQ-16** | **1** |  |  | X |  |  |  | X | X |
| **REQ-17** | **3** |  |  |  |  |  |  |  | X |
| **REQ-18** | **1** |  |  |  | X |  | X |  |  |
| **REQ-19** | **2** | X |  |  |  | X |  |  |  |
| **REQ-20** | **3** |  |  |  |  |  | X | X | X |
| **Total** | **-----------** | **12** | **11** | **12** | **11** | **19** | **10** | **8** | **12** |

**Fully Dressed Description**

UC-5: Information Access

1. User requests a specific chain or job using the Glance UI
2. User starts to type name of the Job or chain into the Glance UI
3. User selects the chain or job they wish to view
4. After selecting a chain the Glance UI will update with a list of records
5. User will select a record or type the name in
6. After selecting the user will be able to view the contents of that record
7. After viewing the specific chain the user can repeat number 3-6 to access another chain or record
8. After selecting a job the use can specifically view the outcome of that job. Additionally all job statuses will be broadcast to the glance web server.

UC-8: Scaling

1. User wants to change the node topology
2. User adds a server to the topology using the given RPM
3. The server connects and subscribes to any chains that are not subscribed to
4. The server joins the cluster and begins caching/computing
5. User removes a server they shut down the machine or node.
6. The topology shifts and upon removal nodes will subscribe to the downed servers information.
7. If the user adds or removes clients only the topology will change and not the computation or the cache

UC-3: Check Status

1. Users who are on glance click on the job the category their check can be found under
2. Glance updates and the user repeats step 1 until the desired check is reached.
3. The user will then be able to see if the check is green(pass) or red(fail)
4. The user can repeat steps 1-3 to see other checks
5. Effort Estimation

Assumption: Let PF (Productivity Factor) be equal to 28 hours per UCP (Use Case Point).

**Unadjusted Actor Weight**

|  |  |  |
| --- | --- | --- |
| Actor | Actor’s Goal | Category |
| User | To access chains, records, and information in the ignite server. See the status of checks from the web server. | Complex |
| Ignite Server | Connect and manage other nodes, run jobs (checks), manage the cache, and handle request from the user | Complex |
| Web Sever | Push results of checks from ignite thin client to Glance | Simple |
| Ignite Thin Client | Receive jobs to run, run the jobs on the server and get the results, return information to the web server | Average |
| Ignite Client | Make SQL requests to the server and allow the user to access and monitor the cluster | Average |
| Market View | Provide data to the ignite cluster about chains and records | Simple |

*UAW (Unadjusted Actor Weight) = 3 \* Complex + 2 \* Average + 1 \* Simple*

Calculations = 2(3) + 2(2) + 2(1)

*UAW* = 12

**Unadjusted Use Case Weight**

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Description** | **Category** |
| UC-1 | A user may want to access Market View data via the ignite platform in the form of a single record or chain | Average |
| UC-2 | A user may want to run specific code from another source on the ignite cluster to check the data within ignite | Complex |
| UC-3 | A user may want to see the status (return) of a job/task in ignite | Average |
| UC-4 | A user may want to connect a client and run code in the cluster at different priority levels | Average |
| UC-5 | A user may want to easily access a record, chain, or job status | Average |
| UC-6 | A user may want to access server information at any time | Average |
| UC-7 | A user may want to access server metrics for health or efficiency analysis | Average |
| UC-8 | A user may want to add more server nodes to make computation faster or allow for more cache space | Complex |

*UCW* (*Unadjusted Use Case W eight*) *U* = 15 · *Complex* + 10 · *Average* + 5 · *Simple*

Calculations: 15(2) + 10(6) + 5(0) = 90

*UUCW* = 90

**Adjusted Use Case Points**

*UUCP* (*Adjusted Use Case Points*) = *UAW* + *UUCW*

Calculations: 12 + 90 = 102

*UUCP* = 102

**Technical Complexity Factor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Technical Factor** | **Description** | **Weight** | **Perceived**  **Complexity** | **Product** |
| T1 | Distributed, web-based system required | 2 | 5 | 2(5) = 10 |
| T2 | User expects exceptional response time, internet connectivity permitting | 2 | 4 | 2(4) = 8 |
| T3 | End user efficiency is important | 2 | 4 | 2(4) = 8 |
| T4 | Easy to change | 2 | 2 | 2(2) = 4 |
| T5 | Scalability is important | 2 | 2 | 2(2) = 4 |
| T6 | Internal processing ranges from light to medium | 2 | 3 | 2(3) = 6 |
| T7 | No unique training required | 1 | 0 | 1(0) = 0 |
| Total = ∑[Wi \* Fi]: | | | | 40 |

TCF (Technical Complexity Factor) = 0.6 + 0.01 Σ[Wi · Fi]

Calculations: 0.6 + 0.01[40] = 1

TCF = 1

**Environmental Complexity Factor**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Environmental Factor** | **Description** | **Weight** | **Perceived**  **Impact** | **Product** |
| E1 | Team familiarity with most of the programming languages to be employed. | 2 | 5 | 2(5) = 10 |
| E2 | Project not entirely object oriented  (Because of web-based components). | 1 | 3 | 1(3) = 3 |
| E3 | Stable requirements expected. | 1 | 5 | 1(5) = 5 |
| E4 | Some familiarity with application. | 2 | 3 | 2(3) = 6 |
| E5 | Team members are highly motivated. | 1 | 4 | 1(4) = 4 |
| E6 | Programming languages of average  difficulty are to be employed. | -1 | 3 | -1(3) = -3 |
| E7 | Beginner familiarity with Apache ignite | 1 | 3 | 1(3) = 3 |
| Total =∑[Wi \* Fi]: | | | | 28 |

ECF (Environmental Complexity Factor) = 1.4 − 0.03 Σ[Wi · Fi]

Calculations: 1.4 − 0.03[28] = 0.56

ECF = 0.56

**Use Case Points**

UCP (Use Case Points) = UUCP · TCF · ECF

Calculations: 102 · 1.0 · 0.56 = 57.12

UCP = 57.12

**Duration Estimate**

Duration = UCP · PF

Calculations: 57.12 · 28 = 1599.36

Duration = 1599.36 ≃ 1599 hours

1. Domain Analysis

**Domain Model**

Components of Domain Model:

- Website Interface (Glance)

- Webserver

- Apache Ignite cluster

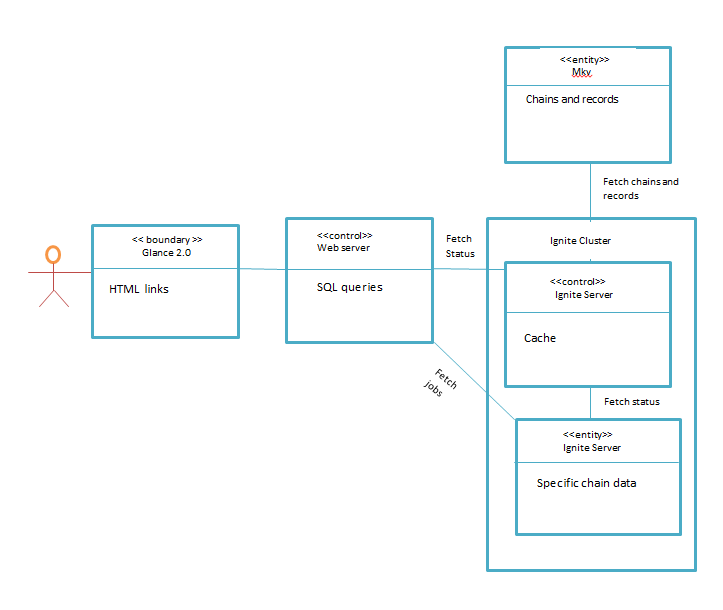
- Job database

- Clients

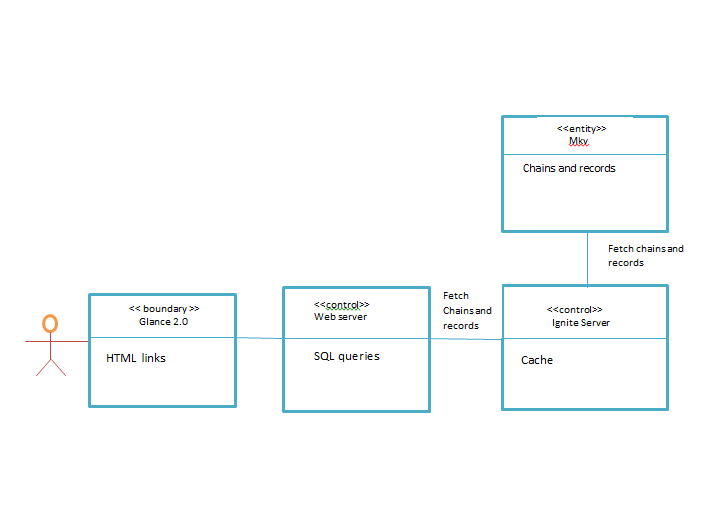
- MarketView database

**Use Cases in Detail**

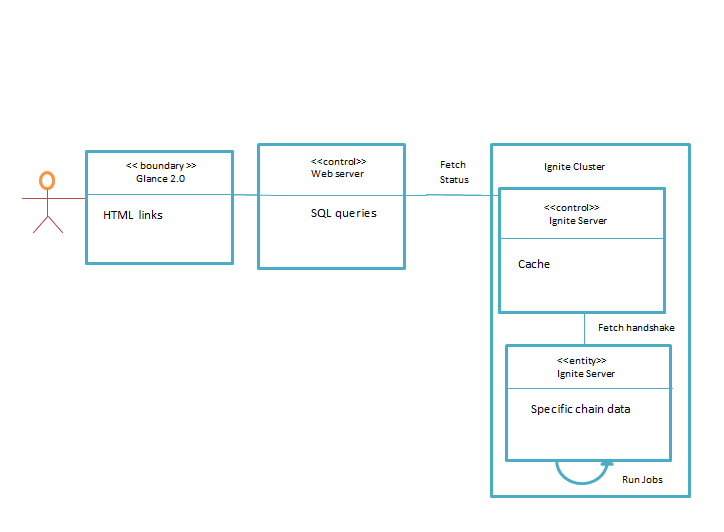
UC-3: Checking Status

****

UC-5: Access chains, records, and checking jobs statuses

****

UC-8: Scaling

****

1. System Architecture and System Design

**Architectural Styles**

We are going to use a multitier architecture design. This design will revolve around the use of a three tier system. Essentially this will separate the UI, Logic, and Data portions of our project. This mimics our intended team layout towards completing the project by breaking down teams to handle each of these modules separately. Each of these modules will be completed on separate platforms; thus ensuring both longevity and upgradability for the future. This approach incorporates the use of multiple platforms for development. This way in the future parts of the project are upgradable or even replaceable. In the case of language upgrades or OS changes.

**Presentation Tier**

User accesses specific chain or job

Checks if chain is subscribed if not, subscribe to the chain

Preform job on server side

Get Chain from Cache

Get Chain info from Market View

Cache information

Manage cluster

Get job from server

Get status from job

Logs metrics

Display job status

**Logic Tier**

**Data Tier**

**Presentation Tier**

This tier represents the UI of the system, including web layout, information display, and check notifications. This portion of the system is the only part of the software where the user can access the program directly. Accordingly, the presentation tier communicates with the other tiers in order to exchange information between the data and logic tier for the user to manipulate indirectly. This tier will handle button presses and the read-only operations needed for Glance to perform. The presentation layer will also handle the notifications for failed checks and problems noticed by jobs within the cluster. Finally, the data from Glance front end needs to be translated into an understandable format; thus, this tier must handle that transformation.

**Logic Tier**

This tier pulls information from the database and does most of the computing and heavy lifting of the program. Mainly, this tier will focus on the calculation of checks, the management of cache, user connections and changes in topology. Along with these calculations this tier will communicate request for information and statuses among other nodes in the cluster and clients. I will act a single middleman between Market View data and Glance, servicing checks, publishing statuses and responding to user queries. The logic tier will handle the communication between cluster nodes. Additionally, this tier handles the scaling features of apache ignite by refreshing topologies and tracking data and health metrics. This tier also handles the dictionary for searching through chains on the user side.

**Data Tier**

Data consists of chains, records, jobs, metrics, and job statuses. Users should not be able to directly access this information without going through the logic tier. In case queries they will be processed on the client and send to the server (Logic Tier) and then the information will filter back through to the user. This tier will provide an api for the logic tier to use to retrieve information. The data tier will store and manipulate data by itself to limit its dependence on the logic tier thus the creation and deletion of data will primarily be in this tier. Additionally, it will handle the inclusion and expulsion of information from the cache for checks, jobs, and queries. In conclusion, this tier will primarily call Market View and handle cache distribution/maintenance.

**Identifying Subsystems**

Our software uses Model-View Controller approach, as well as the Client-Server approach. The chain, and job information is handled by the server machine and the client machine deals with calls to the server as well as data visualization. The server acts as the model and controller and the client acts as the viewer

**Database System (MKV)**

Functions include:

1. Database Management: Stores the hundreds of information about records in relation to their chains.

**Ignite Cluster**

Cache System

This handles the task of compiling the data into a cache system for easy access for the processing system.

1. Job storage: Stores the logic behind the jobs
2. User Database: Stores the subscribed chains’ information.
3. Node Database: Stores node topology
4. Backup: Stores backups from other nodes

**Processing System**

This handles the computation of jobs, and changes in the cluster’s structure, as well as servicing client requests

1. Job Execution: Runs jobs given to it by the web server
2. Node Detection: Computes topology, and adds nodes to system or removes them
3. Request Handler: Handles the client request
4. References

<https://apacheignite.readme.io/v2.5/docs>

<https://ignite.apache.org/releases/latest/javadoc/org/apache/ignite/>

<https://docs.gradle.org/current/userguide/userguide.html>

<http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase7-521261.html>

<https://cedt-confluence.nam.nsroot.net/confluence/display/1571041/Glance>