Project Goals and Scope:

We want to enable a bandwidth / application performance predictor. This will enable the customer to determine if they can deploy a new app to their network, region, and/or multiple sites. All data to perform the calculation exists; we need to pull it and run a calculation on the data.

We want to calculate based on bandwidth utilization, jitter, latency, packet loss SLA across all links being considered.

Results of application prediction is a comparative analysis of the selected ports and sites with graphs showing all selected sites relevant ports –throughput, jitter, packet loss, latency. This is also with a gradient scale indicating the performance acceptability of the application across the sites and the port. The calculation would be run on all exterior facing ports on all sites with the ability to toggle between the port screens as shown at the top of the screen.

As previously stated above, we are looking to build an Application Performance Predictor Engine used within our portal framework for the Telecommunications industry (Data Center, Network Data, Systems, Edge service elements, etc.)

- Build a generic application performance calculator engine (or at least a framework)
- Must be able to input multiple data points / types of data
- Must be capable of mathematical calculations, statistical and comparative analytics
- Must have exposed APIs for connecting to various options (Network Data, OSS/BSS Systems and databases, and into a customer-facing front-end portal)
- Must aim to be focused on Java, Javascript, and .Net, MySQL (open to suggestions) – open source in general from a toolset perspective
- Must be built into a user-friendly dashboard interface (GUI) to prove execution in business format
- Must use open source toolsets and be exposed into a graphic/analytics interface that can integrate into a portal (below are a few screenshots of the architecture and GUI).

Roles:

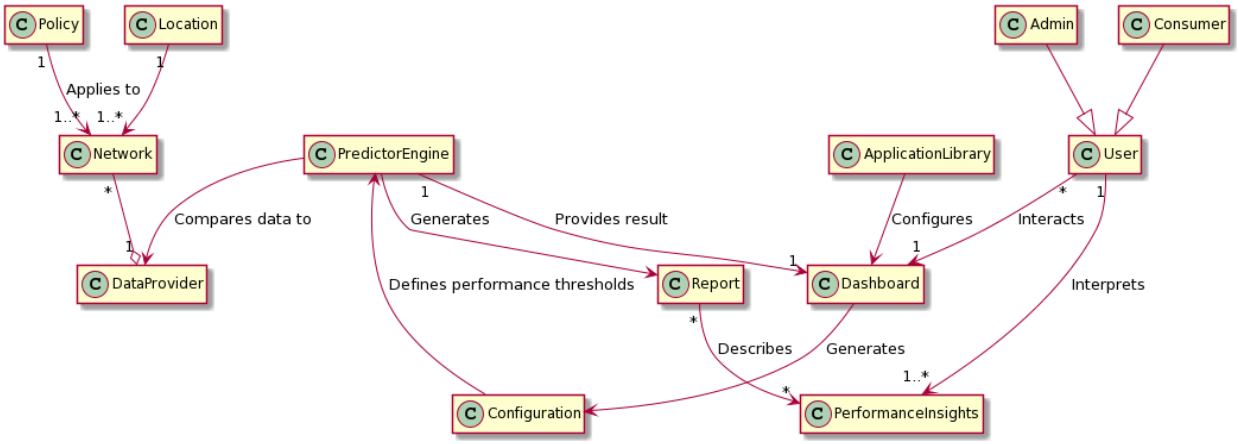
Michael: *Project Manager*
Emma: *Communication Liaison*
Andrew: *Testing Lead*
Shane: *Database Architect*
Matthew: *Frontend Lead*

Project Synopsis:

The CloudSmartz Application Performance Predictor is a web application that will predict the performance viability of a given software application across a variety of different environments and contexts. For a given application and deployment location, the user will see an indication of whether the performance will be optimal, fair, or poor. Data visualization will be accomplished

via a user-friendly dashboard interface. The relevant metrics taken into account are throughput, jitter, packet loss, and latency. Acceptable values of each metric are defined by the related Service Level Agreements (SLA) for a given application. Data collected over a range of 30 days will be used to evaluate the application's performance against the thresholds defined by the SLA for each metric, then against user-defined standards for optimal performance. The Application Performance Predictor Engine will be developed on a backend of Java APIs using the Spring framework, and the data will be stored in a MySQL database. Development of the frontend will use TypeScript with the Angular framework. The project will utilize a CI/CD pipeline supported through Jenkins. The Agile model will be employed to drive the process forward, specifically following the Scrum framework. There will be a variety of both process and project metrics used to track overall progress and quality, such as burndown rate, cyclomatic complexity, and average bug-fix time. On a continuous basis, the sponsor will be updated with progress reports and demos showcasing new features as they are developed. Our process will enable visibility that will allow us to verify our product and work with the sponsor to validate its fulfillment of the requirements.

Domain Model:



Planned Activities & Artifacts:

- We will use an Agile methodology (Scrum + 4-up), implementing two week sprints beginning with a sprint planning meeting and concluding with a sprint retrospective.
- There will be weekly meetings with the sponsor to go over planned scope and report progress, as well as to provide any required artifacts and demo new features.
- At the end of the sprint, the sponsor will be presented with a demo of newly implemented features with the opportunity to give feedback.
- There will also be weekly meetings with the team to go over impediments and general progress of each task.

- The Trello board will be used to manage tasks to complete/ general progress of each task
    - A Trello workflow will be defined that specifies the path a card/story/spike takes through the board (i.e., from product backlog to sprint backlog to development to testing to done).
    - Each task will have a “story point” value assigned to it that will be used to judge the effort needed to complete the task.
- Our shared Google Drive will house documentation, presentations given to sponsors, and meeting notes.

The process methodology that was chosen for this project is iterative in nature. This was due to the constraints of the sponsors, requiring an Agile approach. In addition, this process will allow for continuous improvement of the overall product by allowing us to meet weekly with the sponsor, get updates and clarifications on progress, and update accordingly.

Completing 4-ups will create snapshots of our progress that are easy for the sponsor to understand. Having a well-defined Trello structure with lists for each step of a card’s journey will increase visibility; the sponsor can look at the board at any time and know the exact status of each card.

Standards & Quality Practices:

- Requires at least 2 approvals for a pull request can be merged
- All coding changes must be confirmed to be building properly and have full test coverage before they can be merged into the master branch.
- Design and Architecture patterns will be implemented (when applicable) and documented in depth.
- All written code should be compliant with current industry standards and best practices, including:
    - Proper use of commenting
    - Descriptive variable names
    - Documentation written for complex subsystems
- A certain code coverage threshold (e.g. 80%) will be enforced for new code and for the whole project, if possible. We will discuss this with the sponsor to determine what their

expectations are for code coverage. Coverage reports will be shown during sprint demos, where areas of significantly increased/decreased coverage will be highlighted.
- The team will be performing pair programming whenever possible throughout the project

Tools:

- We will use pull requests (or the equivalent feature in the Git platform used) to facilitate code reviews. Reviewers will leave comments/tasks on the pull request, then use the approval feature when the tasks have been completed.
- We will use a Trello board to develop and track all the needed tasks for the entire scope of the project.
- A CI/CD platform will be provided by the sponsor and will allow us to implement quality checks, automatic test coverage calculations, and general management of the repository, as well as code deployment.

User Stories:

Organizations wants to install apps in their ecosystem and know where to deploy this to be optimal (overall goal)

- One user (not many) wants the capability to see how well they can deploy a new application in a new “place”.
- User wants to know how current bandwidth, latency, jitter, and packet loss will allow a new application deployment.
- User wants to know if deployment will be poor, okay, good, or great.
- User wants to target a particular location to deploy a new application.
- User wants to see the locations available to them in a dropdown.
- User wants a concrete way to see how the network will react to deployment
- User would like to see options next to each other to compare to each other.
- User wants to look at one particular application at a time.
- Business admin wants to configure the engine to set constraints. (low priority)

Process Metrics:

1. **Bi-weekly Burndown:** A chart representing the changes in the amount of outstanding work over time
2. **Velocity:** On average, for a section of previous sprints, how many tasks are getting completed.

3. **Percentage Completed:** The number of completed tasks as it relates to the amount of tasks that are yet to be completed. Will also be useful in predicting future scope creep.

Project Metrics:

1. **Code coverage:** We will use available coverage tools (most likely JaCoCo for the backend and Karma for the frontend) to generate coverage reports that we can share with the sponsor and compare with past reports.
2. **Cyclomatic Complexity:** The number of individual linear paths within a code base.
3. **Percentage of bugs that resurface:** The sponsor requested this metric so that we can assess the extent to which we are learning as well as the strength of our communication and shared understanding with the sponsor
4. **Average time to fix a bug:** The average time between a bug being logged (i.e., being entered in the board) and it being moved to the “done” list. We can track this average over time to evaluate the team’s improvement