K8s Checkmate

Design Document

Team sdmay20-58
Workiva
Julie Rursch
Daniel Brink, Jacob Cram, Sean Sailer,
Alex Stevenson, and John Young
sdmay20-58@iastate.edu
https://sdmay20-58.sd.ece.iastate.edu/

Revised: Final

Executive Summary

Engineering Standards and Design Practices

No hardware, purely software

Agile

Modular code

Clean code

Well documented code

Summary of Requirements

Runs in CLI

Parses Helm charts

Creates policies based on rules provided in templates

checks configs of helm charts and alerts if they fail

Applicable Courses from Iowa State University Curriculum

SE 309 - Working as a team with the Agile work process

SE 363 - Used Docker after the provided virtual machines were not working.

SE 311 - Working with data structures and developing the most efficient algorithms

As far as for the tools that we are using for this project, there are no classes that are teaching extensive Python or Docker skills. For the classes that we did use these tools in, it was only because we were interested in learning and the learning was self guided.

New Skills/Knowledge acquired that was not taught in courses

Configuration with Kubernetes

Parsing Files with Security Checks

Experience with Python Programming Language

Communicating to an Advisor of our group’s progress

Communicating to an Advisor with team management

Manipulation of Helm Charts

Knowledge of Python in General

Helm configuration

Docker usage

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

Julie Rursch - Group Advisor

Eric Anders - Workiva

Thank you for your contributions!

1.2 Problem and Project Statement

There does not exist a tool that checks the configuration of helm charts nor the configurations produced by them. Linters and syntax checkers exist, however, they only check that the helm charts are formatted correctly. They do not check that clusters comply with predefined rule sets.

By creating an extensible framework we hope to provide a well documented, highly extensible, useful tool that prevents a lot of security issues that can exist when using kubernetes. Often the setup of these services is done without enough thought put into the security of the company. Our project will allow companies to be more confident about the security and the correct initialization of their clustered computing setups.

There is nothing out there that exists on what we are trying to accomplish. Our drive is not only to give the open-source community this tool but also be the first people to craft a tool like this. Since this project will be open source it will also service its users better because of the potential for future community development.

1.3 Operational Environment

The end product will run in a Command Line Interface, and will not be exposed to unusually hazardous conditions. This is solely software-based, so there will be no expectations physically for the product. However, we do expect the end product to be able to run on Linux and MacOS.

1.4 Requirements

Functional requirements

- System should parse and check Kubernetes configuration files
- System should parse and check helm charts
- Command-line interface should allow for easy interaction with the system
- System should alert the user of potential security vulnerabilities
- System should suggest how to fix potential security vulnerabilities

Economic requirements

Since the project is almost entirely software, there are very few economic requirements. There is no hardware that needs to be purchased or licenses that need to be paid for since the project will be entirely open-source.

Environmental requirements

Again, since the project is almost entirely software-bound, the physical environment has no effect on it. There is no hardware that could be exposed to the elements or poor weather. In terms of the computer architecture environment, the project should be able to run on macOS and *NIX systems.

UI requirements

To keep the program lightweight and portable, the UI will not consist of a GUI, but rather a command-line interface. This is plenty sufficient for usability and fulfilling the intended use cases.

1.5 Intended Users and Uses

The intended user is the person conducting a security review for a Kubernetes project.

The intended use is to streamline and reduce user error in the process of checking security configurations against a defined ruleset.

1.6 Assumptions and Limitations

Assumptions

We assume that the user will have a basic knowledge of Kubernetes security configurations.

We assume that the user will have a basic knowledge of security with files in general

These assumptions are made because the intended user is for someone who wants to perform security checks with Kubernetes. Someone who doesn't have this previous knowledge probably wouldn't be using this then.

Limitations

The end product will be lightweight and run in a CLI.

We expect the end product to be able to run Linux and macOS.

We expect the end product to be able to effectively perform a security review.

It will not add functionality to kubernetes, but to the initialization and setup.

The product will need to be sought after in order to be found as it is not a commercial product.

1.7 Expected End Product and Deliverables

The end product and deliverables for our senior design project are as follows.

- A lightweight and portable CLI program that can check, verify, and alert users about potentially insecure and vulnerable Kubernetes configurations and helm charts. Lightweight means that it must be a small program that can be downloaded quickly from any internet connection. Portable means that it is not system-dependent and can run on a multitude of Operating Systems.
- Extensive documentation on the installation and use of the program so that anyone will be able to understand and use this. This documentation will include readme markdown documentation for outside users intending to use the program. Additionally, this will include a well-commented code for the open-source community intending to clone and contribute to the repository.
- Open-source code for continual improvement by the open-source community. It is proven that open-source code is more cost-effective, quicker to develop, much more secure/transparent, and more extensible in the future by anyone. We are making our program open-sourced for the aforementioned reasons

The delivery dates for these deliverables is T.B.D due to the nature of the senior design program. We can estimate that the above will be ready sometime around May of 2020.

2. Specifications and Analysis

2.1 Proposed Approach

We have a proof-of-concept program capable of parsing configuration YAML files.

The application will parse and analyze Helm and Kubernetes security configuration files, and compare the results against a defined set of rules.

The application will run in CLI.

The application will accept templates of rule sets to compare to.

The application will be rigorously tested to sufficiently ensure correctness.

2.2 Design Analysis

Our group has been communicating with both our advisor and each other about team roles and planning for our code development. We have mostly been communicating through online messaging with our entire team. Although our team has also met in person as well. So far our meetings, whether that be face-to-face or online, have been very successful. Each one of us is able to understand what is expected and we are able to hold each other accountable for tasks that need to get done.

Our strengths are communication and expectations. Everyone in our team is okay with sharing their thoughts and ideas. Expectations are clearly understood and set as well. Our biggest weakness is availability. All of our members are extremely busy so finding times to meet in person is a challenge.

Observations and thoughts on our team style so far are mostly positive. We all are communicating effectively and getting tasks done on time.

Finally, our team members have been learning the Python language. This is the language that we will be developing our code in.

2.3 Development Process

We are using an Agile development process because our requirements are well-defined but we are meeting with our team every couple of weeks to make adjustments if needed.

We are developing one part at a time, testing it with our tests, showing our advisors what we have how it works now, and testing it in the environment it will be used, and making any necessary changes.

2.4 Conceptual Sketch

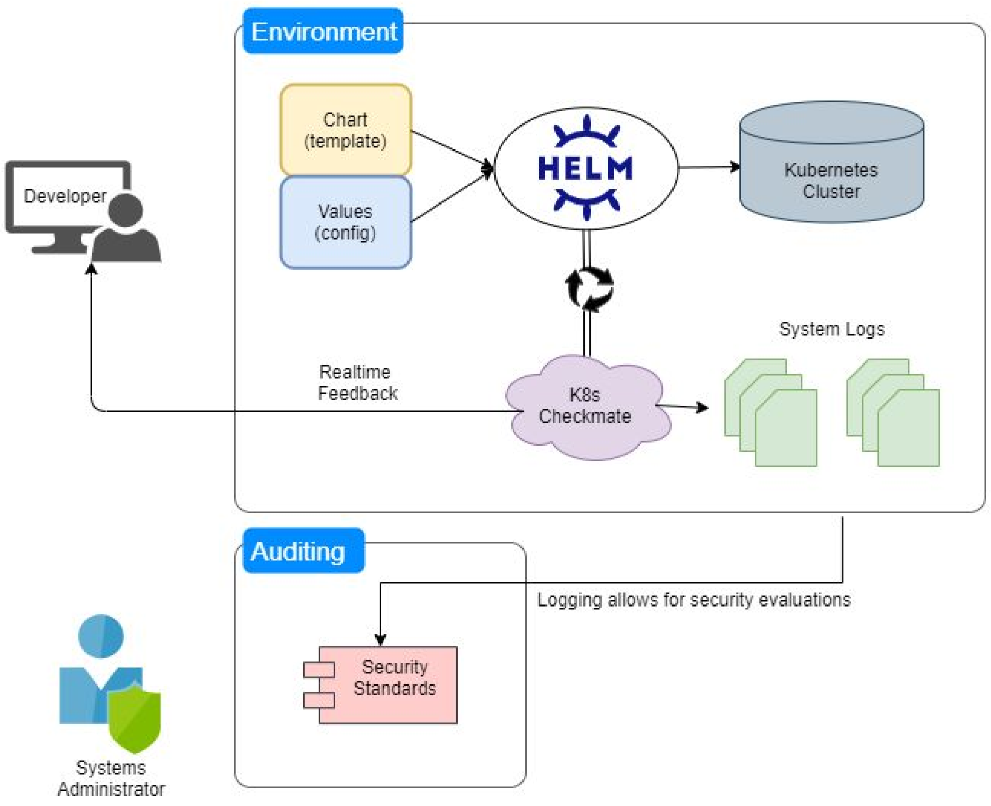


Figure 1: Conceptual Sketch

3. Statement of Work

3.1 Previous Work And Literature

One product that has some similarities to our product is Amazon's "Inspector". This program automatically improves the security of applications on AWS. Then it will show the user where the threats are and explain to the user how serious each threat is. The difference is that "Inspector" only deals with applications on AWS and it doesn't relate to Helm at all.

Source: See *Link 1*

Aqua works with Kubernetes and performs security checks daily. After the check, Aqua will make a report based on their findings. This is similar to our project because this analyzes security flaws with Kubernetes. Although this differs from our project because our project is focusing more heavily on Helm charts.

See Link 2

3.2 Technology Considerations

Strengths -

Hardware wise there wasn't much consideration because our product is purely software. One of our requirements was that our product should be able to run on any OS that supported Kubernetes. So our strengths for these considerations were mostly that there wasn't much to consider on the hardware side.

Languages wise though had a lot of thought. Our group mostly considered the difference between coding in Python and Go. Python was chosen in the end. The strengths of Python were as follows: the language is relaxed when it comes to syntax, it is compatible with Kubernetes, and everyone in our group was very familiar with the language.

Weaknesses -

The main weakness with Python was the testing and importing. Our team struggled with effectively importing different methods to different classes while working with Python. Setting up Pytesting was also a struggle for our team. We were unfamiliar with effectively importing these tools/methods.

Trade-offs -

The benefits far outweighed the struggles our team faced with importing tools/methods. Once our team overcame the weaknesses, we were able to quickly develop our code/tests. The biggest tradeoff was time for faster development overtime.

Had our team stuck with Go, we would've had a harder time coding our parsing functions, storing information, and testing our code. Python's advantage over Go is that Python's generic types with structures and variables were easier to implement than Go was. This was mostly due because our team had more experience with Python than Go.

3.3 Task Decomposition

The main tasks that we will need to do for this are to break the helm chart/kubernetes down before and after running in order to parse them to check for inaccuracies. Once this is done we will be able to create a user interface for the project. This user interface is important because it is what is going to allow us to add the extras after aside from the parsing, we will need to finish that before we go on to making a template generator. Aside from that it is just the linter that will need to be added and whatever else we want to add if we find that we want more.

3.4 Possible Risks And Risk Management

Lack of experience in the area is a risk we are actively combating through studying Python and Kubernetes.

Loss of one or more team members is a possible severe risk, we are mitigating this risk by ensuring our documentation is routinely up-to-date such that a team change would not result in a catastrophic loss of progress or information.

Risks such as the obsolescence of Kubernetes are insignificantly likely, though even in the event such things come to pass we could transfer the skills we learn here to whatever may replace it.

3.5 Project Proposed Milestones and Evaluation Criteria

Key milestones would consist of the following: Parsing of helmcharts/kubernetes files, parse templates and store values after configuration, check values of parsed info to make sure it has finished correctly, create alerts based off of the incorrect info, make a user interface for the application, setup a template generator, add a linter to the system. Each of these milestones are designed so that they are able to be tested task-wise. When we reach these milestones we will know because all of these are provable/tangible parts of our project. The tests for each of these milestones will be dependent on what is being tested. Most of the important testing will be to make sure our parsed information is correct and we will need to spend a lot of time on this because the entire project is reliant on this being correct.

3.6 Project Tracking Procedures

First off we spoke with our advisor and we set milestones on what we want to accomplish. We are going to track ourselves with when we hit those milestones and whether it was before or after our "due date". Also, we will be tracking our progress through completed issues on Github. Based on how many issues get done per person, we will determine the difficulty of every task and determine how much work they have done. So if a GitHub issue is harder than normal, then when that task is completed then that person will have done more work compared to an average Github issue. We are following the agile style of development so we will also make note of when someone gets their smaller tasks done on time.

3.7 Expected Results and Validation

The desired outcome of the project is a lightweight application running on the command line, parsing Kubernetes configurations and alerting the user on incorrect configurations. The application should be able to be configured for various templates.

We will confirm that our solutions work at a high level through rigorous testing and using charts used in the client's environment.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

Dec 30 2019 Phase 1

- Project design planning
- Requirements gathering
- Regular team meetings to plan and schedule
- Familiarization with framework and languages
- Design document drafting
- Initial implementation of the parser

Feb 15 2020 phase 2

- Ability to parse templates AND store proper configuration values

Mar 10 phase 3

- Alert/Gracefully handle misconfigured values

Mar 20 phase 4

- Make a GUI for easy interaction with the program

- GUI is either standalone or integrated with a tool like Rancher

April 1 phase 5

- Provide template generation - making it easier to user and less error prone
- Possibly make a GUI for template generation
- Like phase 4, could integrate with a tool like Rancher

April 12 phase 6:

- Implement functionality to alert on CVEs found in the stack running the containers

April 22 phase 7:

- Add a linter for security policy templates
- Add additional features as we see fit
- Optimize for speed and size

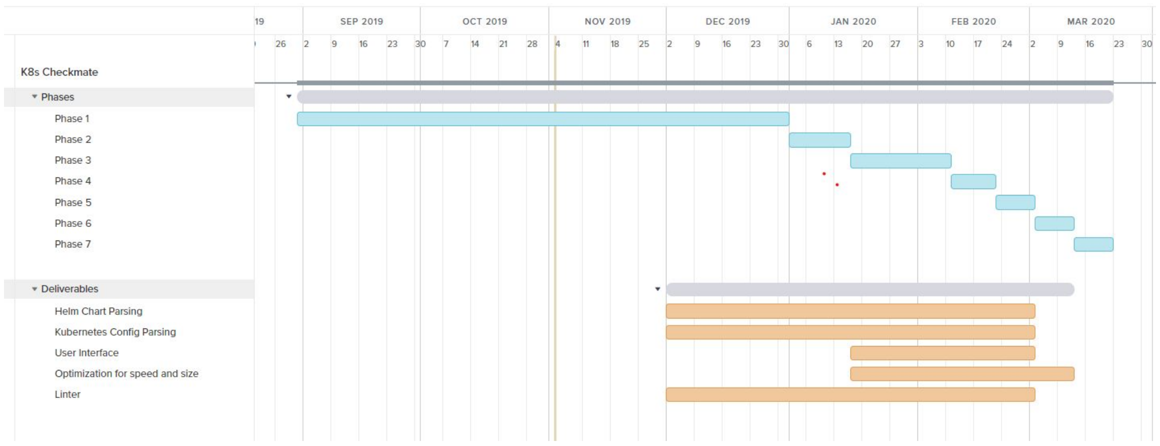


Figure 2: Gantt Chart

4.2 Feasibility Assessment

Realistically, the project will sufficiently fill our requirements, though it is unlikely to fulfill some of our unofficial open-ended nonfunctional requirements. Examples of challenges we have foreseen are that GoLang does not support the data structures we initially planned to use, requiring us to reconfigure our plans and use Python instead, and that because only superficially similar applications exist we cannot take significantly useful inspiration from those.

4.3 Personnel Effort Requirements

Table 1: Effort requirements with explanation

Task	Text reference/explanation	Estimate of effort
Parsing of helm charts/kubernetes files	This will be the first milestone that we will need to accomplish and will allow a base to check against	This is going to be the medium difficulty because it is important that this is robust and holds up with many test cases.
Parse templates and store values after configuration	This will likely be mainly be similar to the previous one but a bit more difficult because they will be checked off of configured setups.	This will be similar to above but with an added layer of difficulty as we are needing to check the already run templates to verify that the setups have run correctly and store that info.
Check values of parsed info to make sure it has finished correctly	This will be the core of our product and will be important to get correct, as well as important to test.	Assuming we are able to get our information setup side by side with the before an after, this will just be a check to make sure the values are the same
Create alerts based off of the incorrect info	This is going to the first part of the user portion of the program	This is going to be triggered by the above check, and should not be
Make a user interface for the application	This will make the application easy to use	This will be difficult because we will need to make sure it is simple to use and difficult to break. Also making this robust for our use will be important so that it eases the use for the customer.
Setup a template generator	This is an addition and not a core feature but will be important	This will, as above help with the robustness for user ease. It will not be difficult, but will require us to be very knowledgeable on the relationships between different settings on the helm charts.
Add a linter to the system	This is another feature that will add robustness to our application	The linter is something that already exists, but since we would like our project to be powerful it is important to have it. Since it does exist, it will not be too hard, and we

		will be able to take inspiration from other open source examples.

4.4 Other Resource Requirements

No additional resources will be required to conduct the project.

4.5 Financial Requirements

No additional financial resources will be required to conduct the project.

5. Testing and Implementation

5.1 Interface Specifications

Our project will not be dealing with hardware and software interfacing with each other and our project will be able to be run on both Unix based systems and Windows systems since it will be written using Python. Because of this, the effects of hardware interfacing will not be important.

5.2 Hardware and software

We do not require any hardware specifications due to our project being a lightweight software program. We are currently testing in visual studio code with simple test cases and simple yaml files to validate our proof of concept.

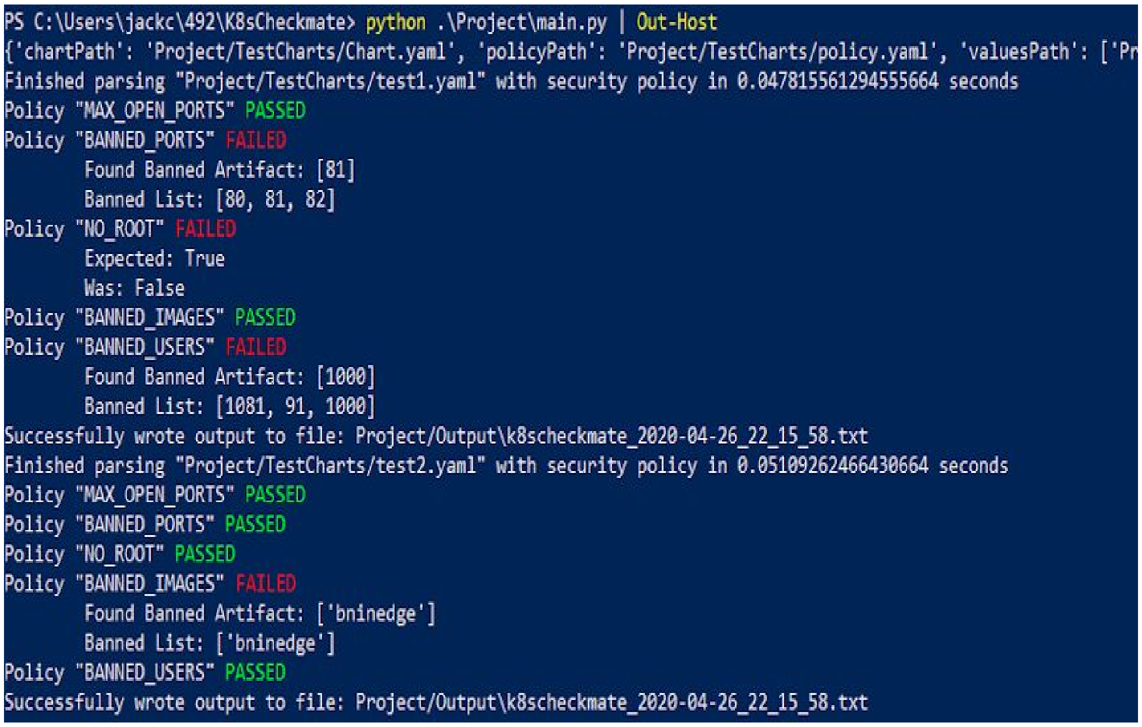
We currently have some basic unit tests to ensure there is no regression in functionality. We initially planned on having much more robust and comprehensive testing implemented. However, changing programming languages and dealing with programming language level issues got in the way of that.

5.3 Functional Testing

In addition to unit testing, we performed functional testing to ensure that we are meeting the requirements laid out by the client. We focus on working with the Chart.yaml files and the Values.yaml files. During our functional testing, we ensured that the project works with:

- Both relative and absolute paths

- A variety of charts with different fields and structures
- Charts with values allowed/not allowed within the defined security policy
- Different security policies
- Multiple values files
- Random charts from the Helm Chart repository



```
PS C:\Users\jackc\492\K8sCheckmate> python .\Project\main.py | Out-Host
{'chartPath': 'Project/TestCharts/Chart.yaml', 'policyPath': 'Project/TestCharts/policy.yaml', 'valuesPath': ['Pr
Finished parsing "Project/TestCharts/test1.yaml" with security policy in 0.047815561294555664 seconds
Policy "MAX_OPEN_PORTS" PASSED
Policy "BANNED_PORTS" FAILED
        Found Banned Artifact: [81]
        Banned List: [80, 81, 82]
Policy "NO_ROOT" FAILED
        Expected: True
        Was: False
Policy "BANNED_IMAGES" PASSED
Policy "BANNED_USERS" FAILED
        Found Banned Artifact: [1000]
        Banned List: [1081, 91, 1000]
Successfully wrote output to file: Project/Output\k8scheckmate_2020-04-26_22_15_58.txt
Finished parsing "Project/TestCharts/test2.yaml" with security policy in 0.05109262466430664 seconds
Policy "MAX_OPEN_PORTS" PASSED
Policy "BANNED_PORTS" PASSED
Policy "NO_ROOT" PASSED
Policy "BANNED_IMAGES" FAILED
        Found Banned Artifact: ['bninedge']
        Banned List: ['bninedge']
Policy "BANNED_USERS" PASSED
Successfully wrote output to file: Project/Output\k8scheckmate_2020-04-26_22_15_58.txt
```

Photo 1: Example of Passed/Failed Policies

A link to a video of our program running during functional testing:
https://drive.google.com/file/d/1v1DxydQfTo82FCzYJgzKHv5UIyMGMDED/view?usp=sharing

5.4 Non-Functional Testing

Performance -

The project must be fast enough that it could reasonably run in a CI/CD system and on a developer's computer. To test the performance of the application, we print how long the program took to run. In the screenshot above, it took about .05 seconds to check each of the values files that were passed in. This is about the average time it takes to run. Given that this speed is orders of magnitude faster than what the maximum acceptable time (a few minutes) would be, not much more testing needs to be done.

Security -

Our final product will be open source so anyone would be able to use it and contribute to the project. We also wanted to make sure we are doing everything in a safe manner to protect not just our client, but all potential users. Our program does not require any special permissions, does not affect the underlying system in any way, and uses a safe loader for the yaml files. Consequently, there is no grounds for any security concerns regarding our project.

Usability -

Usability testing was done by having our client utilize our product. The only aspect that users will have to change would be the policies that they want to check and the file paths to their policy file, value file, chart file, and output directory. That sounds like a lot of paths but we countered that with all those file paths being in one config file that our product will use throughout the parsing/checking.

Compatibility -

We were able to effectively run our program on Windows, MacOS, and Linux. This exceeds the requirement set by the client that it works with MacOS and Linux. The program has also been successfully tested in the client's environment.

5.5 Process

We mocked a simple input/result test. We gave test files that we expected to pass and then we gave test files that we expected to fail. Then if the desired result was not met, we would make changes to our code.

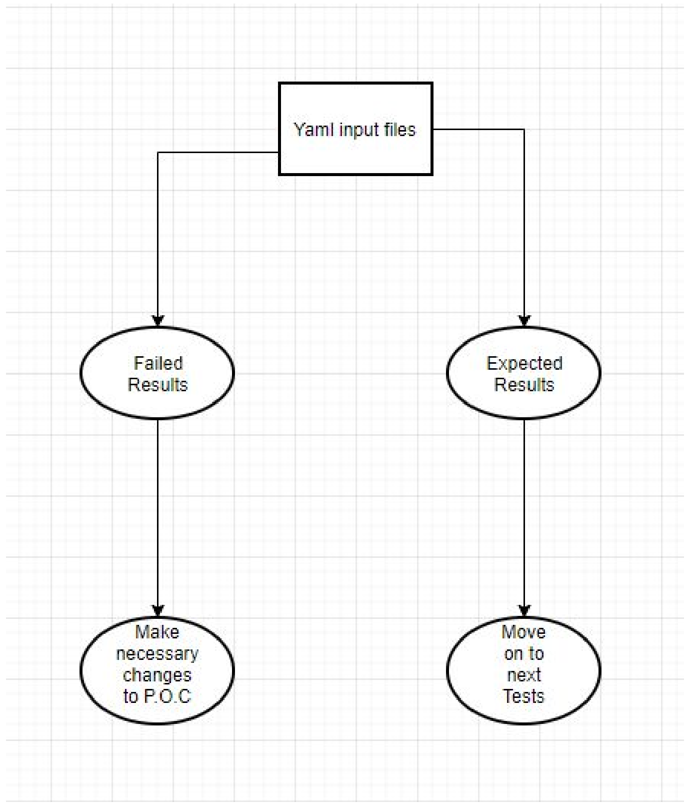


Photo 2: process mockup

5.6 Results

The testing phase was where we spent a bulk of our time.

Failures

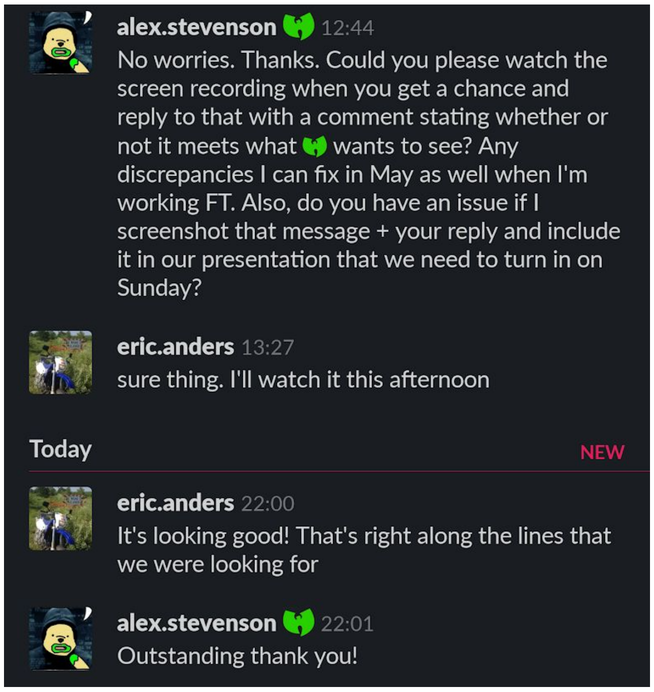
We were having a lot of issues expanding on our Proof of Concept. We tried a number of solutions for reading and storing charts which ultimately failed. There are ways to do this in Go, but we could not seem to get this to work properly. We spent months trying different work arounds to no avail. The traditional method of reading the data that we needed involves saving the data to a struct and working from there. This immediately failed for us due to how unordered the information we are dealing with is. For example a helm chart could have just one image, or it could have multiple images.

Eventually we got the necessary approval to switch the programming language we were using to

Python. We quickly achieved parity with our old proof of concept and solved the issue we faced in Go. Then however, we started running into Python path issues as well as issues setting up the Python modules we developed. It took a lot of trial and error, but we were able to change how we were doing imports, added a setup.py file and we were able to get back on track with our project.

Successes:

Despite all of the issues we faced, we were able to overcome the setbacks we faced and create a successful project. We built software that was exactly what the client expected. Below is a screenshot of Slack messages with our client where he expressed satisfaction in our results.



alex.stevenson 12:44
No worries. Thanks. Could you please watch the screen recording when you get a chance and reply to that with a comment stating whether or not it meets what wants to see? Any discrepancies I can fix in May as well when I'm working FT. Also, do you have an issue if I screenshot that message + your reply and include it in our presentation that we need to turn in on Sunday?

eric.anders 13:27
sure thing. I'll watch it this afternoon

Today NEW

eric.anders 22:00
It's looking good! That's right along the lines that we were looking for

alex.stevenson 22:01
Outstanding thank you!

Photo 3 - Client Approval

In the planning phase of the project, we broke the project up into six phases that would be worked on sequentially. We structured our work so that we met the client's requirements once we completed phase three. This was to allow time to resolve any setbacks that we might face, as well as to try to deliver the best possible project to the client. Everything after phase three was additional features to improve the project. This project management setup ended up being very beneficial in the end. This structure allowed us the time needed to resolve our setbacks and still accomplish phase three, satisfying the client.

We also sought to make this project fast. We have all experienced the need to wait a great deal of time for the software necessary for your work to finish running and wanted to avoid that. Thankfully we did. Our program runs in less than a second and lends itself well to being run automatically on a variety of different projects.

Learned:

The two main lessons we learned are: allocate more time than might be strictly necessary and explore alternative routes to the solution when it feels like you're just spinning the wheels. Allocating extra time by scheduling the requirements to be completed halfway through the project afforded us the breathing room we needed to work through the unplanned issues we inevitably encountered. Getting frustrated one afternoon and recreating our initial proof of concept using Python allowed us to switch the language we were using and to start making real progress on our project.

6. Closing Material

6.1 Conclusion

Our product effectively checks a given set of policies against a values file that defines a chart. Based on the given policies and values, our product will effectively alert the user on passed/failed values that have met the user's given policies. Our goals were to effectively parse and alert the user of passed/failed values based off of a user's desired policies. Our only goal that was not met was alerting the user on how to fix/modify policies that failed.

The plan of action that we took to achieve most of our goals was broken into seven phases. What was required from our client was only the first three phases. Each phase had smaller sub-tasks in which each member of our team would complete every two weeks. We gave ourselves strict deadlines for the first three phases since that's what was expected from our client. We had one team member in charge of parsing, one in charge of storing, one in charge for checking values, and two in charge of testing. When a phase was completed, our team members would combine each other's work and then more testing would be done. The constant two-week cycles and testing allowed our team to manipulate code without suffering a big loss in time.

The plan of action that will take place in alerting the user on how to fix the failed policies will get done with more testing at Workiva. Since this project is open source though, more users outside Workiva can add to the product and the maintenance overtime will keep the project up to date.

This will surpass other solutions being tested because of the amount of developers working on this. Nothing like our product has been made before, but the open source aspect will give the opportunity to have more eyes utilizing our product and refining it. The constant maintenance will ensure that this surpasses other previous tested solutions.

6.2 References

P. Balogh and S. Guba, “Detecting and blocking vulnerable containers in Kubernetes (deployments),” · *Banzai Cloud*. [Online]. Available: https://banzaicloud.com/blog/anchore-image-validation/. [Accessed: 26-Apr-2020].

hub.helm.sh. [Online]. Available: https://hub.helm.sh/charts/banzaicloud-stable/anchore-policy-validator. [Accessed: 26-Apr-2020].

hub.helm.sh. [Online]. Available: https://hub.helm.sh/charts/smallstep/autocert. [Accessed: 26-Apr-2020].

“Single Sign-on SSH & Production Identity,” *Smallstep*. [Online]. Available: https://smallstep.com/. [Accessed: 26-Apr-2020].

6.3 Appendices

Appendix I - Operation Manual

To find step by step instructions on how to setup/demo/test please visit the readme in our repository