K8s Checkmate

DESIGN DOCUMENT

Team 58
Workiva
Julie Rursch
Team Members/Roles
sdmay20-58@iastate.edu
https://sdmay20-58.sd.ece.iastate.edu/

Revised: v2

Executive Summary

Development Standards & Practices Used

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

Docker

According to the Docker documentation, Docker is a platform for developers and sysadmins to develop, deploy, and run applications in containers.

Containerization is not a new concept, but Docker's improvements made it feasible for widespread practical use. At the heart of it, containerization is just another method of virtualization. Traditionally Virtual Machines (VMs) are used for this. VMs virtualize the entire system, from the hardware level to the application level. As a result, they are easier to use on different machines than containers that only virtualize from the operating system (OS) level up.

Docker containers are implemented differently. First, a Docker process is run on the host OS. Containers natively run on Linux through the Docker process; only

virtualizing the apps, binaries, and libraries it needs. All Docker containers are separate from each other, but share the OS kernel with the host OS. As a result, Docker containers are portable, lightweight, stackable, and scalable. A more in-depth explanation is detailed here.

While still relatively young, Docker is a well-established technology. It is open-source, first released in 2013, and has widespread use by companies and developers around the world.

Most 300 level classes in the SE Department provide VMs for students to use, so we do gain some experience with virtualization. Unfortunately, this experience does not translate well to Docker's implementation.

Our project specifically targets Helm charts and Kubernetes. Kubernetes is a technology developed by Google for creating clusters of Docker containers. Kubernetes clusters can be difficult to set up, thus helm charts are a framework used to simplify this process. As Docker is not taught in any of our classes, by extension Kubernetes and Helm are not either.

Kubernetes, Helm, and Docker are all written in the Go programming language. Therefore we will be using Go for our project.

Go

Golang, also known as Go, is a language created by Google to solve a lot of the language level programming issues they were facing at the time. The first stable release of Go was in 2011, and the first major release was in 2012. Go is a fast-growing language that is becoming increasingly popular due to its size and flexibility. Unfortunately, Go is neither taught nor allowed to be used in any of the classes we have taken. However, much like learning any other programming language, the prior programming experience we have learned in the SE curriculum helps reduce the time it takes to learn.

While not explicitly taught any of the technologies we will be using for our project, we will be relying on our skills to learn them. Our skills were honed through the classes we have taken and as a result, we will be better able to teach ourselves what we need to know for our project.

New Skills/Knowledge acquired that was not taught in courses

Configuration with Kubernetes

Parsing Files with Security Checks

Experience with Go 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 (This should be similar to the project plan)

1 Introduction

1.1 Acknowledgment

Julie Rursch - Group Advisor

She meets with us at least once a month in person and communicates with us when needed. She is guiding our group by letting us know what is expected and she lets us know what is expected.

Eric Anders - Workiva

Eric Anders works with Workiva and is our group's Industry Advisor

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.

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.

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 the will be no expectations physically for the product. However, we do expect the end product to be able to run in 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.

Expectations:

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.

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 Design

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.

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 GO 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 of 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 of the necessary changes.

2.4 Design Plan

We are developing our code one feature at a time. This ensures our code is modular. Each feature can run on its own and is imported into the main package. The high degree of modularity makes the code more extensible so that others can modify it for their purposes.

The GUIs are going to be written as local client/servers. This model reduces the overhead and allows them to be run from the terminal, regardless of the desktop environment. This also reduces the number of dependencies needed for the application.

As of now we do not know all of the dependencies and modules we will be using. We anticipate that we will only need modules and dependencies we will have will be from the standard library.

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: https://aws.amazon.com/inspector/

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.

https://www.aquasec.com/solutions/kubernetes-container-security/

3.2 Technology Considerations

Our project is entirely software so technology considerations will only be in the computer area. As for the language that we are going to use, we are going to code in GO.

Some of the strengths of using GO are that it's compile based, has memory safety, and has a garbage collection.

Weakness: Most of our team hasn't used GO before so we had to learn the language

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 GoLang 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 give reach these milestones we will know because all of these are provable/tangible parts of our project. In fact, these are basically where the same times that we need to do major testing on each portion. 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 use in Kubernetes environments.

4. Project Timeline, Estimated Resources, and Challenges

4.1 Project Timeline

Febuary 2020 - Phase 3 done

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

Jan 15 2020 phase 2

- Ability to parse templates AND store proper configuration values

Feb 10 phase 3

- Alert/Gracefully handle misconfigured values

Feb 20 phase 4

- Make a GUI for easy interaction with the program
- GUI is either standalone or integrated with a tool like Rancher

March 1 phase 5

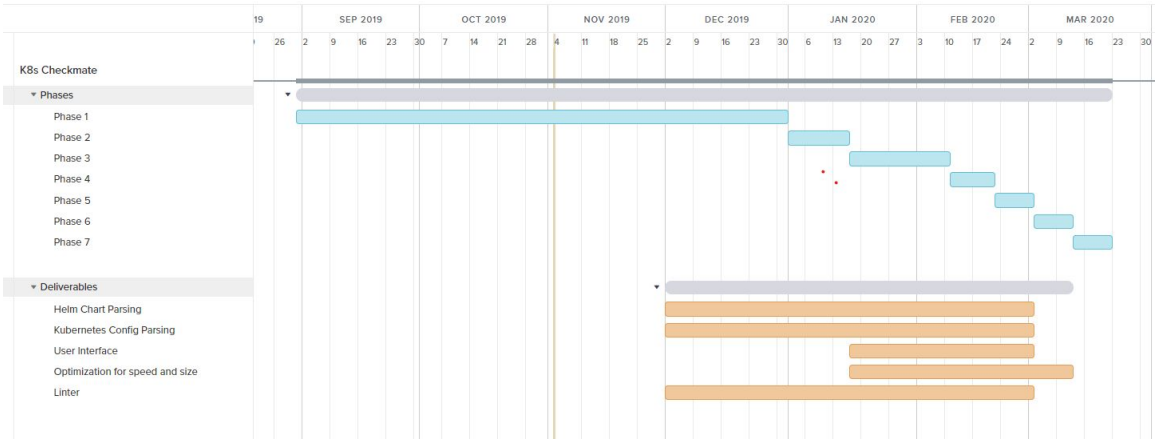
- Provide template generation - making it easier to user and less error prone
- Possibly make a GUI for template generation
- Like phase 4, could integrate with a tool like Rancher

March 12 phase 6:

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

March 22 phase 7:

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



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 that because only superficially similar applications exist we cannot take significantly useful inspiration from those.

4.3 Personnel Effort Requirements

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

Testing is an **extremely** important component of most projects, whether it involves a circuit, a process, or a software library

Although the tooling is usually significantly different, the testing process is typically quite similar regardless of CprE, EE, or SE themed project:

1. Define the needed types of tests (unit testing for modules, integrity testing for interfaces, user-study for functional and non-functional requirements)
2. Define the individual items to be tested
3. Define, design, and develop the actual test cases
4. Determine the anticipated test results for each test case 5. Perform the actual tests

6. Evaluate the actual test results
7. Make the necessary changes to the product being tested 8. Perform any necessary retesting
9. Document the entire testing process and its results

Include Functional and Non-Functional Testing, Modeling and Simulations, challenges you've determined.

5.1 Interface Specifications

– Discuss any hardware/software interfacing that you are working on for testing your project

5.2 Hardware and software

– Indicate any hardware and/or software used in the testing phase
– Provide brief, simple introductions for each to explain the usefulness of each

5.3 Functional Testing

Examples include unit, integration, system, acceptance testing

5.4 Non-Functional Testing

Testing for performance, security, usability, compatibility

5.5 Process

– Explain how each method indicated in Section 2 was tested
– Flow diagram of the process if applicable (should be for most projects)

5.6 Results

– List and explain any and all results obtained so far during the testing phase

- – Include failures and successes
- – Explain what you learned and how you are planning to change it as you progress with your project
- – If you are including figures, please include captions and cite it in the text

• This part will likely need to be refined in your 492 semester where the majority of the implementation and testing work will take place

-**Modeling and Simulation**: This could be logic analyzation, waveform outputs, block testing. 3D model renders, modeling graphs.

-List the **implementation Issues and Challenges**.

6. Closing Material

6.1 Conclusion

Summarize the work you have done so far. Briefly re-iterate your goals. Then, re-iterate the best plan of action (or solution) to achieving your goals and indicate why this surpasses all other possible solutions tested.

6.2 References

This will likely be different than in project plan, since these will be technical references versus related work / market survey references. Do professional citation style(ex. IEEE).

6.3 Appendices

Any additional information that would be helpful to the evaluation of your design document.

If you have any large graphs, tables, or similar that does not directly pertain to the problem but helps support it, include that here. This would also be a good area to include hardware/software manuals used. May include CAD files, circuit schematics, layout etc. PCB testing issues etc. Software bugs etc.