ACIT5900

MASTER THESIS

in

Applied Computer and Information Technology

(ACIT)

May 2022

Cloud Based Services and Operations

Towards the Canary Manager

Exploring a high-level language for automation of canary deployment

Odin Kanstad Grimstvedt

Department of Computer Science

Faculty of Technology, Art and Design

Abstract

In this submission, we deal with themes and topics related to software deployment, but more specifically around the deployment strategy known as canary deployment, and the possibilities of automation within the concept. This thesis will explore surrounding facets of canaries, how research represents the method and how canaries are viewed in literature.

With this submission, the goal is to propose an approach to handling canary deployments through a high-level language. To achieve a relevant contribution towards this goal, an abstract model containing some of the envisioned key features of the language is created. With the model, the potential for reducing the complexity of implementing canary deployment as a strategy to a project is showing promise by reducing the lengthiness of scripts, centralizing the command of the deployment, and providing a syntax with a high degree of human readability.

The model is then tested on a canary deployment case described in the works of a preceding thesis discussing a similar topic, which ends up suggesting the need for a high level language. This is done to validate the assumed advantages of employing a high-level language rather than a series of tools or scripts. Furthermore, the model language's expressional power is tested by analysing its ability to respond to different scenarios with varying needs.

As a result, the language reaches a level of a framework, which shows promise as an improved alternative to combinations of tools and plugins for achieving canary deployment, as well as transcending its initial desired function, by introducing additional implementational features that extends the language's abilities.

Contents

List of Figures

Acknowledgements

I would like to extend my profound and sincerest thanks for my counselor, Kyrre Begnum, and my admiration for his ability to involve himself in the material produced, cooperating on ideas regardless of their level of quality,and his capacity for encouragement, understanding and conveying wisdom.

An acknowledgement and deep appreciation for my better half, Iselin Kvernevik, for her endless support, assistance and efforts in motivating me, despite being over encumbered herself by her own masters and work while simultaneously preparing for our household to gain a new constituent.

Additionally, I wish to thank my family for their support in the process, and their understanding of my being apparently ceasing to exist for the better half of a year.

Lastly, gratitude must also be expressed towards my dog, Trude, for forcing me out of out my habitat regardless of the situation, providing me with much needed exercise and ventilation.

Chapter 1

Introduction

The release of the first commercial computer, the UNIVAC, led to a ever-expanding industry of software development, which is in a continuous state of evolution and adaptation. As the industry has reached different aspects of societies, and the current paradigm of the software development industry is characterized by a desire to quickly respond to requests and a need for adaptability in the face of constant changes in the landscape. To achieve these limber traits within an organization, a multitude of strategies are employed by companies large and small. Some rely on methodologies and mindsets meant to increase the flexibility of their workflow and introduce differing degrees of responsiveness to their modus operandi, others turn to tools and techniques such as development patterns for effectivization of their production line. More often than not, a bit of both are implemented to achieve a competitive level of this sought-after industrial flexibility.

Birthed from the agile notion and the technologies spawned in its wake, a wide array of techniques designed to bring the work of the software developers out to the target audience or intended user group. Deployment strategies, which will be one of the main concerns of this document, are ways of taking a digital ecosystem out of its incubator and bringing it to it's desired habitat, all the while ensuring that it is healthy and functions as intended. Though there are constant strides being made within this domain of deploying software, with tools and heuristics guiding the process, moving complex sets of software components to a new location by deployment can still be rather demanding, and in some cases proves to be an arduous task. The topic of this thesis will be to reduce this strain of deploying such shipments of code, by attempting to advance towards automation within the field. To achieve this, propositions for a syntactic structure will be presented, with a strategy known as Canary deployment, serving as the motivation for carving out the foundation of the automation language. This concept will be detailed later in the document, but first a presentation of the working problem statements of this thesis:

- P1: Investigate potential language features, implementational approaches and tools to ascertain the feasibility of automatic canary deployment.
- P2: Produce a conceptual framework for a language with the expressive power necessary to respond to canary deployment scenarios

An *investigation* into the topic of canary deployment, existing languages, and tools that may share the same goal or orientation as the topic of this thesis will hopefully yield an understanding of how to approach the task. Also, identifying the core principles and idioms within the sphere of canary deployment will reveal what capabilities the language needs, and charting the tools that revolve around the same concept might provide either support or inspiration in designing a language.

Within the realm of software development, there exists multiple paradigms of programming languages, with their own subgroups holding different values, strengths and features. Establishing a syntactic structure for something that potentially has no counterpart may involve treading an untrodden path, but lending from the strengths of a language or paradigm will surely prove to effectivize the process.

When taking on the challenge of designing a language, there are undoubtedly a wide multitude of angles one can take to work towards a goal. Therefore, an *implementation approach* will be decided upon to guide the following design process towards its destination.

Upon encroaching the topic of deployment, one will undoubtedly encounter different systems and *tools* which are designed to aid in the process of rolling out systems. While investigating, discerning whether these tools may either be used to alleviate the workload of producing this language, or as lessons on what to do and possibly what to avoid.

A *conceptual framework* consisting of core features and structural principles will grant the possibility of employing the design on cases detailing a *canary deployment scenario*. This will allow for further understanding of the language's needs, and subsequently additional development.

Working towards *expressive power* means widening the breadth or range of capabilities within the language. As the deployment scenarios' grow in complexity, so does the need for the languages' ability to represent solutions to the different challenges within the scenario.

Chapter 2

Background

In this section, the topics deemed necessary to discuss automation of software deployment will be discussed. A detailing of the state of the software industry will be given, explain how the market places requirements and demands on organizations and its developers and why this should be of ones concern. Furthermore, descriptions of the concept automation as well as the deployment strategy central to this thesis will be given before attempting to bring insight as to what existing research there is revolving the theme of this document.

2.0.1 The Pressured Developer

The pace at which new software-centered innovations are released into society is rapidly increasing, with new services and problem solutions being introduced both to the private and industrial sector in an ever-increasing manner. For the companies and actors subsisting on delivering software-based services, the evolving and expanding market is equally an opportunity for business, as it is a potential stressor for the developers. This stressor being a demand for the ability to respond to the market, as their customers are constantly offered new and potentially better solutions to their needs, tempting them away from the products they currently subscribe to. Simultaneously, the corporations need to keep up with the advancing technologies, and incorporate new knowledge into their products as the need arises. In other words, to be able to compete within this industry, one needs to be capable of responding to a continuous demand for improvement and alterations of ones product for it to not be “outrun” by competitors.

This setting generates a tremendous amount of pressure on the engineers and developers responsible for the product and its features, as they are required to not only alter existing-, and produce new features to evolve their product, but also making them available to their users in a state that is satisfactory as to not cause downtime with potential major consequences for the user, or harm the reputation and subsequently the subscriber base of their product. One could argue that responding to rapid innovation by developing competing features and functionalities is a large enough task in and of its own, so by adding another dimension of responsibility with the management of deployments and all that it entails, could be considered an overestimation of how much an individual or group should be able to manage.

Another way to look at it, is to disregard the arguments about how demanding the field of work is, and just examine where the working hours are spent. Time spent on implementing the expansions or alterations of a product, is time not spent on evolving and creating new improvements. Should one therefore manage to remove, or at least alleviate the need for developers to shift their focus between tasks, the resulting freed up time could be spent on the work that yields the most profit, or makes the service most competitive.

As this thesis harbors the assumption that the software developer is in a position of increasing strain being put on them by various sources, an attempt to justify this assumption, and why it is relevant for this thesis’ aim will promptly be made.

2.0.2 Yerkes-Dodson's Law

Nan and Harter draws upon Yerkes-Dodson's law when discussing the squeeze developers experience when being pulled between responsibilities and demands put on them. (Ning Nan Harter, 2009). The theorized law, developed within the domain of psychology, describes a relation between performance and arousal, or task demand as Nan and Harter titles it. In this model, a curve depicting the level of performance according to the anxiety, i.e. pressure put on an individual in relation to performing in a situation. The graph, as seen below, implies that there is an optimum area which exists in between a highly strenuous and demanding task, and a trivial piece of work, where the middle ground leads to the best levels of performance for the members of the group set to solve the challenge.

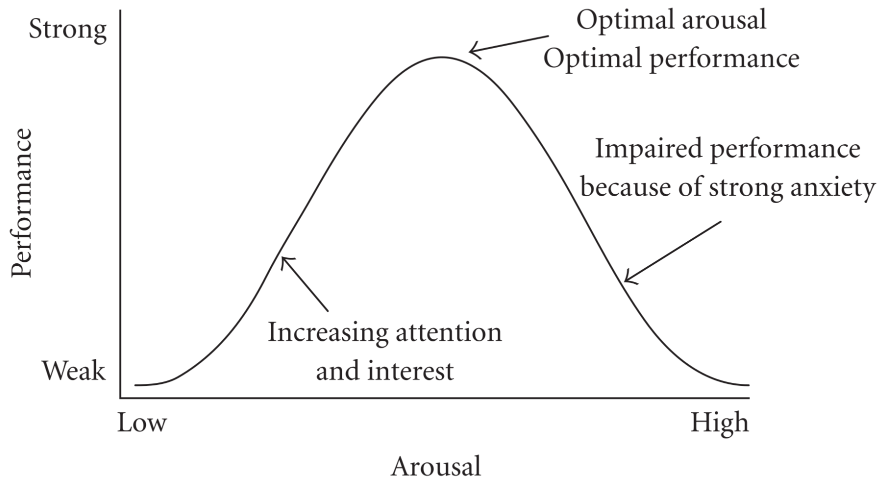


Figure 2.1: A graph depicting Yerkes-Dodson's Law

Should one trust the validity of the model, it serves as a good tool to illustrate one of the points of this paper, which is the outcome of having too much pressure put on the developers. If the assumption of developers being put to an over encumbered state workload-wise is true, one can expect to find them as outliers on the right-hand side of the graph, where performance levels are meager and impaired. Finding a metric to accurately position the discussed group on the graph has not been done, but following the assumption in this this paper, they are to be placed on the downward slope, and should today's trend continue, one can argue that they will be found increasingly towards the right-hand side. Some might thrive in high pressure environments, but there is reason in expecting a too high level of demand on employees will cause the quality of production to be negatively affected by this. To counteract the pull towards the overly strained and subsequently poor performance of developers being overloaded with responsibility, this document will suggest alleviating parts of their workload through means of automation.

2.0.3 Automation

"Automation research emphasizes efficiency, productivity, quality, and reliability, focusing on systems that operate autonomously, often in structured environments over extended periods, and on the explicit structuring of such environments" (Goldberg, 2012).

With this statement, Goldberg submits the main focal areas considered when researching automation, which can in turn be interpreted into a definition of automation:
A process turning a system autonomous, where efficiency and productivity are valued without a tapering of the reliability or quality of the performing system.

This is achieved by employing technology to handle tasks, with a lessened level of human assistance. It is a concept existing within many fields where one can expect to find monotonous chores, both physically as well as digitally. Within the realm of information technology, there are many aspects which are reliant on automation, with a growing range of areas it can be applicable to. The idea automation is widespread both on the macro, as well as on the microscopic scale with a swathe of processes tied to industry, machinery, robotics, hardware and software, but the most relevant area for this thesis is within software development life cycle.

One of the hotter automation topics in this field, is automated testing, at least according to search engine results when exploring the topic of automation. Tools developed to alleviate the drudging work that it can be to design, run and interpret the result of a myriad of tests for all the components making up a service. This is one example where software developers have been unburdened by recognizing the patterns of the work needed in the software development life cycle, and turned the previously manual chores into automatic processes. One could interpret this more recent surge of interest for automated testing as it being part of the work which was previously considered too intricate to be effectively automated, but due to technological advances is no longer within this category. Arguably, there is still a vast amount of the current responsibilities of a software developer which ought to be automated, and in due time we will likely see much more of their manual processes taken over by clever automation software, giving the developer a bit more refuge to seek out the coffee-dispenser without a sense of guilt following them from their desk.

Why the processes waiting to be automated are still done manually might relate to what areas are feasibly automatable (Goldberg, 2012). Automation excels where the scope of the function outputs is predictable or the range of desired outcomes is well known. On the other hand, one of the challenges of automation arises when the task at hand becomes too complex and intricate, and where decisions of a less straightforward nature must be made. This document insinuates that the deployment strategy known as canary deployment is now one of the topics that are within the realm of the attainable for automation.

2.0.4 Canary Deployment

Having shone light on the desired state of autonomy, a reasonable follow-up would be a clarification of what this thesis aims to have automated. Though the concept might be a familiar one for most, a quick description of what a deployment is, before presenting what a Canary Deployment strategy consists of is warranted.

A software deployment method or strategy within is a set of activities rendering the system or service accessible for its intended users (Carzaniga et al., 1998). These activities does not simply entail installation of new additions, but may include processes such as user training, and integration of new features (de Andrade et al., 2021, p. 1683). The act of deploying can include a variety of activities, and a varying range of steps needed to take the product from one environment, region or phase, to another desired endpoint. To overcome the different needs posed by deploying software, a wide variety of approaches have been developed, all of which provide their user with different features, and have different requirements to function. This submission centres around one of these strategies, specifically the deployment method known as ”Canary Deployment”.

As the naming scheme suggests, a Canary deployment shares its name with a bird, which has had a peculiar usage in human industry (Singham, 1998). The bird was brought into coal mines, serving a function of security for the miners, as due to it’s size and biological composition would quickly succumb to elevated levels of carbon-dioxide in the tunnels, acting as a warning sign for those in the vicinity of a leakage which could prove deadly. Though not being a matter of life or death, the digital edition of the canary serves the same purpose.

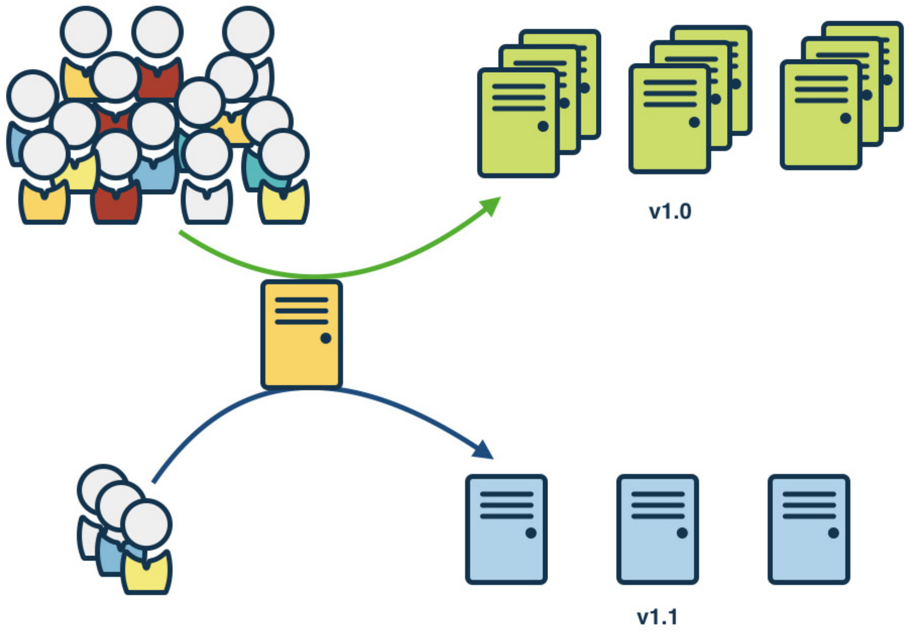


Figure 2.2: A visualization of a Canary deployment
(Implementation Techniques for Canary Releases | GoCD Blog, 2021)

In the image above, a simplified explanation of the Canary deployment concept is visualized. As a new version of a service is ready to be published, it is instantiated within a separate container or encapsulation from the currently running production environment. Through the use of a traffic distribution tool such as a load-balancer, traffic is gradually introduced to the service containing the update. As figure 2.2 illustrates, the initial portion of traffic introduced is a limited size of the total users or requests, whereas the remaining majority is still accessing the predecessor of the update. The newly instantiated canary is continuously measuring its performance compared to a standard specified by metrics from the previous version, or by developer-defined criteria for success. Should the canary environment appear to be failing in some sense, the deployment process can be terminated, rolling back the traffic to the original, tested and safe production environment. And this is where the deployment strategy is borrowing from its similarly named avian mining tool, as the amount of traffic directed towards the newest version is only a small portion of the whole, the amount of damage is minimized, as the remaining traffic has not been exposed to the faulty service before the system administration can pull back from the deployment. In other words, should traffic be equivalent to users, then only a handful of the users have been experiencing issues, while the rest are accessing the service as intended.

2.0.5 Continuity-as-Code

Though some of the motivation for the development of the system presented in this thesis comes from the aforementioned and subsequent sections, one of the more important factors is the groundwork that was laid out in the research of Çeliku, where the proposition of fully automating a Canary deployment is postulated (Çeliku, 2020). In her thesis she coins the term ”Continuity-as-Code”, taking the idea of infrastructure-as-code a step further in the direction of continuous delivery by working towards reducing the complexity of-, and complexity behind canary deployments. The master thesis tours a series of four prototypes, a tool growing in fidelity for each step, until the point where the fourth and final design takes an alternate course, steering the project away from a tool-oriented design, to proposing the idea of a high-level language to manage and deploy a canary strategy.

In her thesis she categorizes her work as exploratory research, justified by both the open-minded and open-ended approach to the topic, as well as due to the lack of research pertaining to canary deployment (Çeliku, 2020, p. 38). Despite having cultivated a research-heavy paper, her work treads into a investigative and experimental realm as there are no comparable systems to what she intends to design.

As a goal for the design, she specifies containerized environments, and orchestration tools as being central to her proposed system, this as it is envisioned existing in a cloud based architecture. As the goal for her work, Çeliku states that her solution will exist within a cloud environment, and will explore the possibilities of automation within containerized, cloud based setup. The strategy to achieve this included the use of orchestration tools such as Kubernetes, and revolve around containerized environments which would render her able to achieve a high degree of fidelity in the prototypes produced, and also reproduce a level of realism, or ”in vivo” as she describes it. Developing a system with cloud orientation in mind seems to a choice aligning with the todays trends, as the amount of enterprises employing cloud services has climbed well above ninety percent according to data from Cisco, interpreted by Sumina in his article pertaining to cloud statistics (Sumina, 2021).

Upon completing the research project, Çeliku speculates in different directions her design would require further development, and among the topics she specifies in future development, a need for a high-level language is deemed necessary to operate the design she presents.

2.0.6 Canary Deployment as a Research Topic

As this paper has something akin to a predecessor, it seems reasonable to identify how it described its similar topic within research, either from industry and academia. In her thesis, Çeliku initializes her study on previous research of canary deployment by making use of the Google trends tool to gain an overview of the relevance of the deployment method via the frequency of look ups on the term (Çeliku, 2021, p. 19).

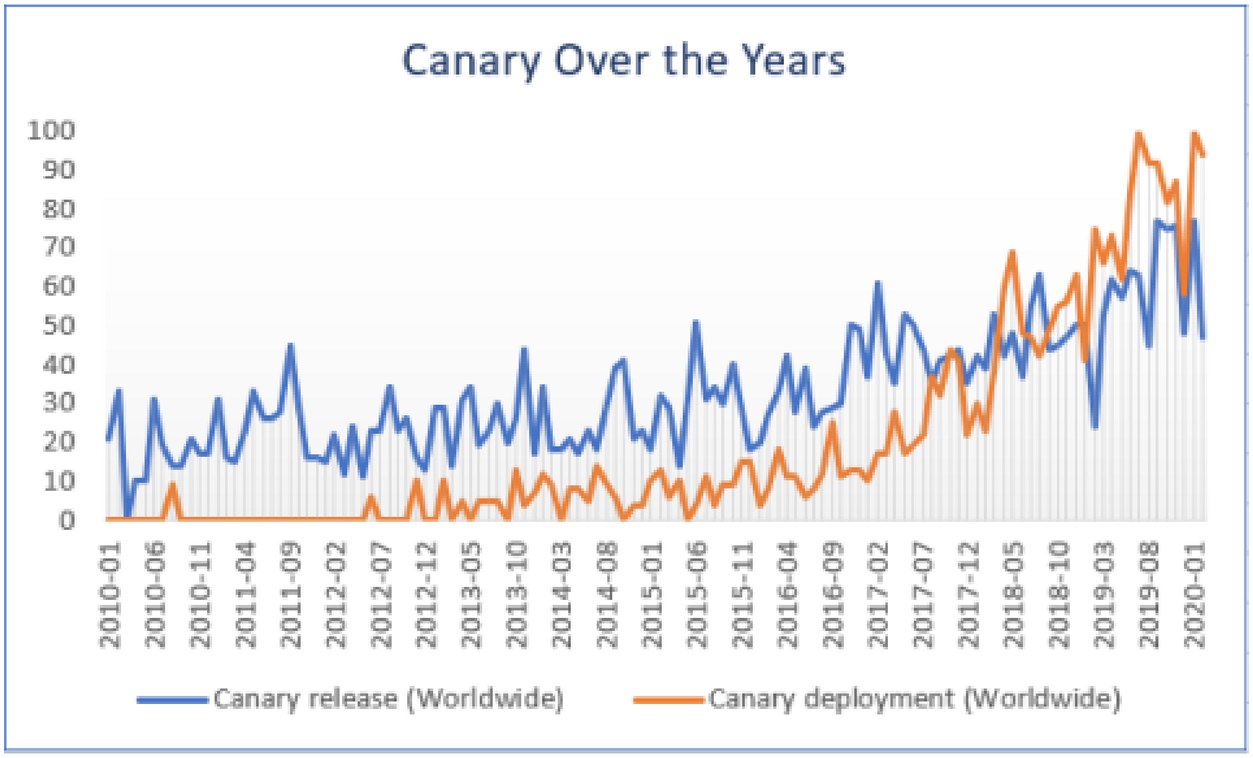


Figure 2.3: Frequency of searches for Canary Deployment and Canary Releases from 2010-2020 (Çeliku, 2020, p. 19)

A quick and reductive interpretation of the statistics seen in the figure above 2.3, implies a recent and occurring growth in the popularity of- or the curiosity towards Canaries. Marking the beginning of the graph, is what Çeliku has uncovered as being the first time a software tool used the name ”Canary”, as well as the release of Google’s stable but experimental version of their web browser, the Chrome Canary (Çeliku, 2020, p. 16). In the search, Google provides the frequencies of searches for both the term Canary release, as well as Canary deployment, of which the latter seems to be the one gaining the most traction in the most recent time, and according to the graph is experiencing a more exponential growth than the similarly named term. It is important to note, that these statistics is in and of itself no indicator of the popularity of Canaries in research, but is based on the number of everyday usage of google to inquire about the terms (FAQ about Google Trends Data, 2022). Meaning, this is representative of the global, common interest for the keywords used in the search. Since her thesis was submitted, two years have passed and the graph now has grown longer. Applying the same keywords to the same tool reveals how the seemingly increasing interest for the canary concept has evolved over the years following 2020.

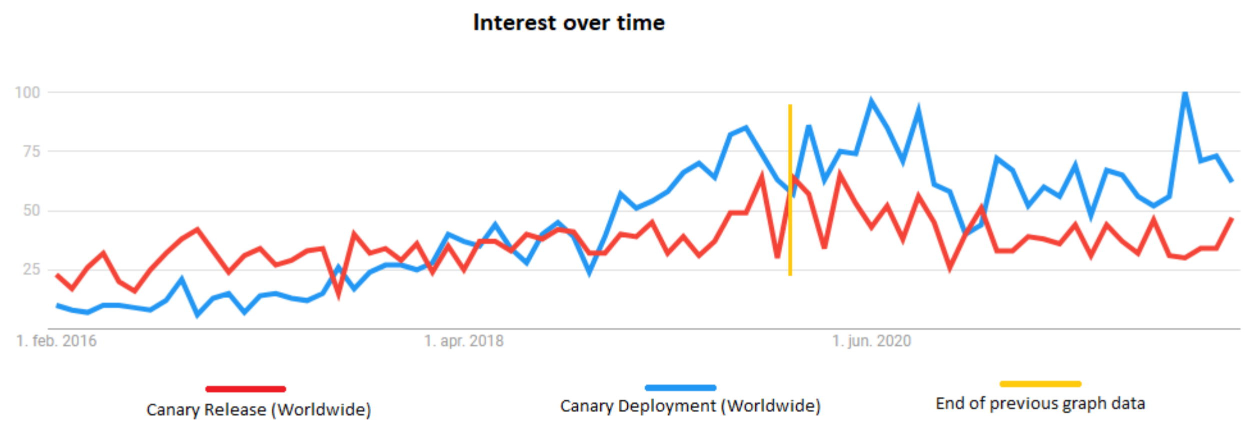


Figure 2.4: Frequency of searches for Canary Deployment and Canary Releases from 2010-2020 (Google Trends, 2022)

Though a surge of interest was occurring at the time of writing, the seemingly growing interest stagnated after the submitting of her thesis. This can be see in the graph above, showing the frequencies of the similar searches spanning from 2017, to may 2022. Though some spikes in interest are clearly occurring, the average inquiry after canaries is apparently no longer increasing.

This notion of canary deployment not making any waves in the development landscape can be experienced by attempting to discern its role through mentions in published articles, either academic, within the industry, or forum where both conglomerate. Upon beginning the investigation, one of the first meetings with an indicator for canary as a deployment strategy being overlooked, was in an article discussing deployment technologies. In this article, subtitled ”A systematic review of deployment automation technologies”, the researchers gather information on the most popular technologies for deployment management, with names such as Puppet, Chef and Ansible reigning at the top of the total 13 tools they present (Wurster et al., 2019). The article goes on to present the features of deployment the different tools provide, arguably in a somewhat high degree of detail, and include discussions of their usage and viability in different scenarios. What makes this article relevant is that there is a total absence of canary mentions, even though there is such a wide list of discussed technologies, all pertaining to deployment tools. Though canaries are not in and of itself a technology, it is a strategy which in theory can be employed alongside many of the available deployment technologies discussed. Therefore, the argument is posed that should the topic of canaries had gained any traction, the minimum of at least a mention alongside these technologies would be expected.

In spite of the lack of mentions in places where one could expect them was seemingly occurring, further investigating the phenomenon was done through another set of channels. The LISA (Large Installation System Administration) conference, is one of the larger and longer running entries in the list of conferences held by the non-profit organization Usenix, known for organizing conventions and for disseminating research (About USENIX, 2017). In their call for participants, the conference encourages submissions that improves upon the day-to-day work, improves their day-to-day, the tech industry as a whole as well as articles that ”demonstrate the present and future of computer systems engineering in all of its forms”, which arguably gives an apt explanation of the diversity of content one can expect to find in their program (LISA21 Call for Participation, 2021). This conference was chosen as the subject of research to better understand the occurrence of canary in research, due to its diversity in submissions, as well as the large amount of entries in their program.

By viewing the conference program in an expanded mode, one gets all the proceedings in any of the tracks available at the conference with not only the title of the entry, but a the following introduction, in the format of an abstract. This expanding on the topics would also prove useful as the titling of talks or papers might be misleading or somewhat unrelated to the actual contents it was introducing. Then, by searching for the keyword ”canary” on these expanded programs, a trawl was done over a series of LISA conferences going back to, and including 2015. The selected year, 2015, is chosen as it marks the start of the upward trend for the interest in the term ”canary deployment”, as can be seen in figure 2.3, which would indicate a larger chance of success in finding material. The following table presents the findings of the search across the conferences.

Year	Findings	Canary being main subject	Topic(s)
2021	No Mention	-	-
2020	No Conference	-	-
2019	1	Yes	Automation of Canary Deployment Evaluation
2018	No Mention	-	-
2017	1	No	Capacity and stability patterns
2016	No Mention	-	-
2015	No Mention	-	-

Data from: (All Conferences | USENIX, 2022)

What the table above shows, is the total amount of mentions of canaries in the program of the five years preceding the writing year of this thesis. As one can see, there are only two entries with canaries involved, with one of the submissions only mentioning it as one of a series of bullet-points. The resulting number of occurrences of canary in LISA conferences can be interpreted as further strengthening of the statements by both Çeliku, as well as Wurster and his co-authors; who brazenly state that there is no academical literature on deployment technology, exclaiming they had to turn to search engines to find details from industry-based sources regarding the tools they were discussing (Wurster et al., 2019, p. 64). That being said, it is important to note that it is understood that this is only one segment of the entire scene where discussions around canary technology and deployment might occur, and is not wholly representative of the research available on the topic.

2.0.7 What does a Canary Deployment involve?

Following the presentation of the meager finds throughout the LISA conventions, of the two entries, the oldest did not represent research in depth presentation of what a canary deployment involves. The most recent entry on the other hand, has a topic which suggest a high degree of relevance for the project at hand, as the submission revolves around automating aspects of a canary deployment. The proceeding from 2019 is a presentation titled ”Enabling Invisible Infrastructure Upgrades With Automated Canary Analysis”; a slideshow presented by a self-proclaimed DevOps enthusiast who was presenting the strategy as a solution to some of the problems developers and system administrators were facing in the industry (McKenna, 2019). The presentation did little in discussing possible use for canaries, alternatives or go in depth in research, so measuring its relevance to the previous section seemed superfluous. However, in his presentation, a series of prerequisites, definitions and statements about canaries are given, which will be used to further understand what a canary deployment currently is depicted to entail.

To dissect the information that is dispensed during the presentation, its contents are represented here, sorted based on the main takeaways.

What Canaries Are

As presented, a Canary is simply a comparison; a measurement of difference between two environments, mainly a production and a development environment (McKenna, 2019). McKenna describes the canary analysis as a tool, but in a manner that could imply that the canary itself is also to be considered a tool i.e. not a concept. He suggest that despite a notion residing withing the IT-community of canaries being complex and unintelligible, it is in reality just a series of data points being measured and evaluated. He continues with expounding that canaries are there to help us automate the most mundane tasks such as migration, and to introduce normalization so that the same process can be applied to large numbers of similar systems.

What Canaries Require

In this presentation, implementing a canary solution is introduced as something that has a non-trivial amount of requirements for it to be functional. Among these, he states a need for an already present CI/CD pipeline, warranting separate environments and work-flow orchestration. Furthermore, a call for a database containing time-series metrics, an environment for executing custom code, and a self-sustaining canary ”judge” software to actuate the evaluations. Based on these prerequisites, the canary is to some extent, only a gear in a series making up a complex machinery, and is presented as having some major dependencies.

What Challenges do Canaries Present

Throughout the entry, McKenna highligths what is needed in terms of organizational change, and postulates that one must prepare to gain buy-in at the highest level possible to be able to restructure the development line in order to employ a canary technique. Furthermore, he proposes that this has an overall effort that would require one year of work to implement. This as there are no off-the-shelf solutions to automated canary evaluations, there is still a need for knowledge about the system as it requires manual attention throughout the process and once again, as the organization would require a functioning CI/CD configuration of the development life-cycle, there might be a need for additional implementations to meet this requirement.

What does a Canary Provide?

According to McKenna, the Canary strategy provides the organization utilizing it with an ”invisible” method of integrating updates (McKenna, 2019). He explains that the use of ”invisible” does not mean fully automatic, and the implementation still necessitates human interaction and some times evaluation, but it streamlines the process. Furthermore, he explains the different type of user, ranging from neurotic development managers not accepting downtime estimates below 99,99%, to their opposites taking joy in test new features in the production environment. This is done as to point out that the metric-reading ability of a canary suits either user, allowing them to decide whether the canary is a success or not depending on their own definition of successful readings.

To summarize the presentation, the canary is a tool centered around evaluations between testing and product environments, which alleviate the developers from the work following an incident occurring at the release of a system modification. The canary itself is described as not too complex, but also as having a not too wide feature range. According to McKenna, implementing the canary requires not only knowledge and an adequate existing architecture, but also potential organizational restructuring with good-will from the upper management of the product owners. Though comments on the contents of this presentation are many, a discussion on this will be left to the section dedicated to discussing, as to not commit sacrilege on the academical foundation this thesis strives to adhere to.

Chapter 3

Approach

As the title suggest, this chapter will present the angle of attack towards the problem statements, and what comprises the method of addressing the topic. A detailing of the goals of the project and their range, a project outline and the methodologies used to convey the results of the design phase will be brought to light.

3.0.1 Method

In the background chapter it is stated that the topic that this thesis pertains to, is rather lacking in comparable research. What this project is attempting to do, is contribute towards creating a new technology, in a sphere where literature on comparable work is unobtainable, and existing material or tools do not provide the solution to the problem space being investigated. Paving the way in uncharted scientific territory means conducting exploratory research, meaning its purpose is to develop ideas and techniques (Elman et al., 2020). Due to the nature of exploratory research, it becomes harder to establish preconceived milestones for the progression of the project, and requires adaptability from the ones conducting the research. This as throughout the unfolding of the product, discoveries and realizations are prone to occur, causing the charted route to tangent around them towards a new course. However, it also leaves a lot of room for choices, which provides the upcoming design phase with a lot of flexibility.

3.0.2 Goals

In the related work by Lea Çeliku, which to some extent laid the foundation for this submission, she postulates the need for a conglomeration of further research trajectories, each concerned with a different necessary aspect of a system capable of handling fully autonomous canary deployment. Her transition from designing a tool, to the realization of the potential advantages residing with solving the problem with a high level language, set the premises for the objectives for this project. In this paper, the objective is to respond to the proposed need for a high-level language by creating a language framework capable of satisfying the two problem statements presented in the introductory section:

- P1: Investigate potential language features, implementational approaches and tools to ascertain the feasibility of automatic canary deployment.
- P2: Produce a conceptual framework for a language with the expressive power necessary to respond to canary deployment scenarios

To achieve these goals, a complete and functioning language is not the requirement, but rather a model encapsulating the most integral aspects of the envisioned language, with a fidelity high enough to be applied to a deployment situation for demonstration purposes.

3.0.3 Project Outline

Going into the design phase, a division of two concretely separated working periods were imagined. The first would entail uncovering the central elements of a canary deployment process, and understand their position and context within the language. With the now ascertained knowledge of what must be included, the second phase would begin, where implementation features would be created, and design choices would be made, both while adhering to the needs uncovered from the previous segment. Albeit the two stages' relatively clear separation of functionality, due to the nature of the exploratory study, the line between them was expected to be somewhat opaque, and sometimes be traversed in both directions.

3.0.4 Methodology

To convey the results emerging from the design process, a selection of methods is employed. As a large part of the work done to carve out the canary manager is done through visualization of requirements, so will the method of presenting it. As the different aspects of the project is discussed, illustrations and graphs depicting abstractions of the functionality or structure will accompany the presentation. The use of pseudo-code will also play a vital role in portraying the intended use of the system, as the actual programmatic structure is not implemented, but rather theorized. That being said, the level of fidelity of the occurrences of pseudo code is varying between illustrative purposes, and being concrete suggestions to a syntax implementation, containing enough detail to express what the envisioned language should be capable of. A comparative analysis between the manager and its predecessor will also be put forth, to showcase what has been achieved and how it solves the case in a hopefully more simplistic and effective manner for the user.

Chapter 4

Results

In this chapter, the resulting design of the system, now having been titled ”Canary Manager” will be presented across four sections, each detailing a different aspect of the design process, in varying degree of abstraction and conceivability.

4.1 Design Assumptions

This section will present the ideas that is to be considered foundational for the proposed system of this thesis. Definitions of core concepts surrounding the topic will be discussed, as well as ideals and principles that lay the foundation for progressing through the design's phases. Here, discussions around terminology and components are presented, and thoughts revolving how they should be handled, and what needs to be considered is aired.

4.1.1 Simplicity

One of the more central values held throughout the designing process of the system was to aim for high degree of simplicity wherever the end user would interact with the language. This would include both simplicity in the form of length; shortening the span of commands and compound statements, as well as aiming for readability and using implicitly understood phrasings when invoking operations, commands and utilizing the language.

To achieve this, the thought of lending from the more popular and widespread programming languages has its allure, as it aids in intuitive use, or ”readily transferred existing skills” as described in Kaltenbacher’s article on human computer interaction (Kaltenbacher, 2017, p. 84). That being said, the purpose of this exploratory study is not to erect a complete language, therefore a freedom is taken, allowing oneself to take inspiration or lend from various languages where it seems suitable, and when it seems to yield the most readable, and intuitive language. The enjoyment of having created an effective interaction format is not the sole reasoning behind desiring a language characterized by simplicity. A claim by authors discussing the topic of simplicity on the conference of human computer interaction, states that the usability of the software is decisive for its success, and thus the simplicity of the user interaction becomes key (Uflacker & Busse, 2022). Meaning, working towards simplicity translates to working towards a desired competitive ability for the language, not only for the creative satisfaction of reducing the lines of code.

4.1.2 System Independence

A motivator and ideal for the design of the Canary Manager is the thought of having the system functioning on any platform, in tandem with any suite of tools or services. Having the manager independent of the architecture resides within will likely yield more usage of the system and allow for a broader user base. It would possibly also render the finished system more durable to change set on by technological advances, as it has no dependencies which also must be maintained or replaced. Albeit being a desired attribute of the system, it is possible that working towards a independent system might lead to a much more complex implementation, and hinder work such as future prototyping.

4.1.3 Language Paradigms

This paper's goal, which was expanded upon in the previous chapter, will not concern itself with the exact syntactic structure of the proposed language, nor does it aspire to strictly adhere to principles held within any of the different factions of programming languages. This as the design process will be built on its own central values and goals. However, in the opening section of Ambler, Burnett and Zimmerman's article on differing programming paradigms, it is stated that languages rarely conform to only one paradigm; instead picking liberally from several to accommodate it's own intended features (Ambler et al., 1992). This fits in well with the assumed method of establishing a design, as the process will involve lending from wherever seems to suit the language best. That being said, the design being presented will lend a greater deal from one specific paradigm than any other.

Object Oriented Programming

There are no limits to the possible combinations of design patterns, paradigms and languages to take inspiration from when discussing the ways of creating a language meant to achieve automation of canary deployment. Considering what wing of IT deployment resides within, one can tend to be biased towards the declarative approaches, which most system administrators are familiar and comfortable with. However, during the early stages of this project's design phase, the question of what is being designed, was accompanied by an inquiry as to who this design is intended for. Should the design ”stay in its lane”, and adhere to the standards of DevOps programmers and their declarative ways, take a procedural route, or follow a different route?

Within the current state of software development, one of the more widespread models for transferring abstract ideas into working software is the concept of object oriented programming (OOP), at least according to forums, online learning platforms and other non-academic sources such as Freecodecamp (Thanoshan MV, 2019). The paradigm revolves around breaking down the desired result into bite-sized objects, turning the complex structure into easily solvable problem-components which in turn will compose the entirety of the system by interconnecting them. Having the program-structure be made up of individual, potentially independent units, allows for a great level of modularity, and opens up for later additions to be made without having to tear down or rebuild similical structural beams of the system to introduce expansions.

As the canary manager would aim to reduce the complexity presented to the user, a high degree of abstraction would be needed. Additionally, one of the reasons for a potentially high level of complexity was due to the looming challenge of huge systems comprised of a myriad of tools and services. To address these potential issues, the way object-oriented thinking breaks down compound problems into solveable, bite-sized units, alongside its ability to abstract digital issues into real-world problems, seemed to fit the bill. Thus, the design will move away from the traditionally declarative of configuration management, and attempt to introduce a different technique which is object orientation.

4.1.4 Domain Specific Language

In every region of software development, there exists a set of concepts and idioms that are central to discussing the topic. These concepts are often abstractions of system components or functionalities, rendering otherwise highly complex sets of machine interactions and engineering marvels more tangible. To ground the language in a realm familiar to those who are envisioned as its future users, some of the more protruding concepts and abstractions within Canary deployment are used to better understand what the language needs to have dominion over. The next section will divulge some of the core concepts making up a Canary, and how they play a part in the design of this language. Before disclosing the entities considered in the language creation, a debrief on how they were selected.

Ensuring that the commands of the syntax are as intuitive and applicable to the domain of the canary manager will aid in making the language, hopefully assisting in reaching the overlying goal of simplicity. Hence, basing the wording of entities and features on relevant and related idioms becomes a process worth investing time into. Achieving this relatability meant understanding the realm one was operating within, and to ground the language in a lingo familiar to those who are envisioned as its future users, some of the more protruding concepts and abstractions within canary deployment should be ascertained to better understand what the language needs to have dominion over.

Compiling a set of articles pertaining to canary and software deployment was gathered, and were entered into a word frequency counter. The list created was stripped of stop-words, case specific names and terms, and the resulting set of words were inspected. A method like this might have proven to be even more effective as a preliminary study, as some of the more pertinent concepts were already conceptualized preceding this search, however, it still provided an overview of options when considering naming schemes and idioms appropriate for the topic.

Literature used in compiling word frequency lists

Title	Keywords	Year
Bifrost: Supporting Continuous Deployment with Automated Enactment of Multi-Phase Live Testing Strategies	Release Engineering, Continuous Deployment, Canary Releases, A/B Testing, Microservices	2016
Towards continuity-as-code from local solutions to a high-level approach for automated canary deployments	Canary Deployment, Continuity, Continous Deployment, Release Engineering, Automation	2020
Rapid canary assessment through proxying and two-stage load balancing	Software Engineering,Software testing, Load Balancing , Continuous Deployment, Canary Releasing	2019
Comparison of zero downtime based deployment techniques in public cloud infrastructure	Component, Cloud computing, Containerization, Orchestration, Microservices, Blue Green strategy, Canary strategy, Zero downtime, Rapid deployments	2020

The table above shows the literature used to compile the selection of wording. As shown through their associated keywords, all of them are involving Canary deployment in one form or another, and center around topics such as release engineering, deployment strategies and continuity one way or another. Based on the keywords, they are deemed to all exist within the realm relevant to this paper, and with the oldest entry in the list being published in 2016, the language used can be seen as relatively up to date.

4.1.5 Canary Concepts

Having acquired a way to secure the selection of language entities' relevance to the topic, the following section will define the terms and their importance to the language.

Environments

In Merriam-Websters' dictionary, an environment is defined as being ”The circumstances, objects, or conditions by which one is surrounded”, which to some extent depicts what an environment in the software-world entails. But to further ground the term in our sphere of operations, a definition by Reiss characterizes a software environment both as the collective suite of tools as well as the whole surrounding software development process (Reiss, 1996). When discussing an environment in relation to deployment, one is referring to all the components making up the totality of a service or application, the sum total of the individual states of these components and how each entity is intertwined, depending or related to another. A simpler simile to liken the (software) environment to the ecosystem of a biome; with all its living entities, its climate and how this single biome is comprised of a multitude of entities, and can be described by having characteristics as a result of the state of its components and surroundings. Upon a deployment, one wishes to either alter the state of a selection of component(s), the relation between entities in the environment, or to introduce and/or remove.

One of the standards many organizations and corporations adhere to, is having multiple, similar but separated environments which each serves a different purpose. A common example would be having one environment dedicated to the development, experimenting and furthering a service, while the other serves as a stable environment which is available to the end user, and is only changed when the alterations introduced are thoroughly tested. This example method of having two environments could be extended further to include several additional replicas of the original environment, should the need arise. It is not uncommon practice to have an environment dedicated to experimenting on the service's features, another for training purposes, one for readying and finalizing updates and lastly a stable production environment which is the only one being accessed by the user. This method of ”cloning” and separating environments allows for the developers to try and fail on alterations and implementations, without having to tiptoe around the inner workings of a service in fear of rendering it unavailable to the end user. Though this separation of concern allows for a lot more freedom, contains the extent of damage and reduces the the overall cost of mishaps, it also introduces the issue of having to replicate the alterations made to one environment onto another. Meaning, without the proper tools or protocols, the workload doubles as any changes to one environment must also be done to the other.

This becomes highly relevant for a canary deployment strategy, as it usually involves the version from development overtaking the one in production. However, it is unsure whether direct references to these environments will be required for the Canary Manager, but keeping it on the forefront of entities which needs consideration seems intuitive.

Traffic

In the event of deploying a service, at some point in the process, the implemented service will be made available to a specific type of user or introduced to a form of activity. In a scenario where a development team elevates their testing environment to a production environment, the previously secluded environment will in most cases now experience a large increase in incoming activity from end users now able to reach it. This interaction of requests being made to the service, and the outgoing responses is dubbed traffic.

To Canaries, one of the most apparent characteristics is the idea of controlling, limiting and distributing traffic flow in a manner or another. By taking the reins of the traffic flow, and gradually increasing the portion of traffic directed to the newest environment, one creates the ability to discern the success and stability of the alterations made to the previous production-level environment on a scale dictated by the operator. With a gradual or incremental approach such as this, the strategies will have a variance on the desired amount of steps and traffic-volume increase, depending on variables such as e.g. the size of the update or what (sub)systems will be affected, and thus the need for having the ability to detail the control of traffic arises. To exemplify, should a hypothetical deployment be of little importance or impact, there could be little to no need for a set of incremental steps. On the other hand, with a more critical update, or a release riddled with uncertainty, modelling the the increments with a higher amount of detail, and with a increased number of steps might be desirable. These differing situations are arguments for the inclusion of traffic as a main syntactic component in what will be later proposed as a draft for the language.

Time

As the deployment progresses from one state to the other, and increases the overtaking of the newer version, the existence of a factor controlling the rate of this progression is somewhat implicit. The need for managing the speed of which the stages of a Canary deployment are introduced will vary depending on the scope of the task, but having the steps progress as soon as all installations are complete, i.e. in the tempo of the processors clock rate, would arguably be a rarely desired format. The concept of Canary deployment is to have an early warning of signs of trouble with the newly released system version, allowing the developers to pull back from the procedure without the service taking too much damage from any mishaps. In scenarios where the deployment is faulty and the service is rendered inaccessible, an immediate rollback would be the correct response, but there exists a swathe of other cases where errors, bugs, or other unforeseen issues leads to a bad reception of the deployment, which would require the service to actually see some run-time for them to be uncovered. Additionally, some measurements require the metrics to have time to establish that their performance is below a desired state. It is because of these situations that having a say of the time allotted for increments and the deployment readily at hand would be a necessary factor. One could remove the need for controlling the intervals duration by hard wiring the system to wait for certain thresholds to be reached, arguably lowering the complexity of the resulting language, but this would most likely result in a system unusable for some, with a lower level of control at the price of sleek design. Therefore, a unit describing the length of each increment, or detailing the progression rate even further to individual steps would most likely prove to be the option rendering the system with the highest usability for a wider range of users.

Conditionals and Operators

Aside from the somewhat obvious arithmetic, relational and most common logical operators, the language of the Canary Manager may have a need for more advanced and specialized operators. It is possible the most frequent usage of the manager will be able to function with a rather simple set of operators, and one can save development time by cutting down on the intricacy of the operators available. The operations the system will presumably see the most frequently, will be measurements of a state-value against a threshold, or a Boolean test, applied as conditionals for the success or failure of the Canary. But as these comparisons grow more complex, there lies a lot of potential in the work of producing advanced operator behaviour, in terms of cutting down code length for the end user. As some ideas which will be presented in a following section, have already been through some conceptual iterations, it was revealed that the Canary Manager has its own set of frequently used statements, which would best serve the user should they be easily employable, and intuitive.

Metrics

As our sensory system's reach does not include the machines we operate, we become incapable of viewing the deployment process as it unfolds. Therefore we become reliant on defining the observable micro-universe that exists within our environments through means such as metrics. Metrics can be readouts of minute details within a service or system, as well as being a sum representing the state of an entire environment. These metrics are engineered by us to produce a view into the conditions of the machines and systems we employ, and therefore become essential in the task of software deployment.

To gauge the process, performance and potential success of the deployment, the system would need to produce relevant and understandable health indicators before, throughout and after the process of deploying. There are several key performance measurements that seem obvious to include, such as the ability of the service to respond to requests, or technical soundness in general, but depending on the service or system being deployed, the metrics one would be interested in could vary greatly. Depending on who one is asking, and what role or responsibility they hold in relation to the service being deployed, different aspects will be of the highest importance. e.g. the users' reception of the service versus the technical performance of an update. Regardless of the intent of the measurements, the system will need to be able to produce and interpret measurements in forms such as Boolean variables, in ways of simple tests as well as an evaluation of a subset of metrics.

Additionally, access to, understanding of, and the ability to manipulate numerical and possibly even text based variables will be necessary.

As per how to gain access to these data, there are different ways to look at who is to be responsible for producing these metrics. One approach would be to have the responsibility lie with the systems administration to not only produce these metrics, but also to convey them to an area or directory where they converge and become available for the system to read. On the other hand, there exists an array of metrics that could be considered common (e.g. processing power usage, traffic flow, memory readings) which should be considered included by default. Another angle of attack is to somehow achieve a method of referencing the components of the environment, and through the manager access the data necessary for the evaluations.

Regardless of approach, if the metrics were to be stored in a designated area, with either the filename, or metadata conveying an identifier, these will quite easily become referable throughout the system, and in turn make the process of creating a deployment schema simplified. A suggestion of this in use can be seen in the section for the Evaluation component. To further add to the range of usability for the system, another dimension is added to the use of metrics. As the canary progresses, and the traffic is increased, it is not unfathomable that one could be interested in a more detailed image of how the new environment is performing under the increased stress. Accessing data from not only the current stable environment, but the previous stages of the canary might be information valuable to some, and increase the number of ways one can discern the performance of the Canary.

Evaluations

Given that a system is able to produce a variety of metrics, a potential hurdle is met when discussing how to interpret them, and more so, ”who” should be responsible for their interpretation. As the language envisioned is independent of platforms and the systems involved are to be trivial, the variety of scenarios for the language’s use is seemingly indefinite. Attempting to standardize the evaluation of a deployments success might render the system inflexible, as the variation in use cases and their goals could prove to large to premeditate and correctly include in an ontology. Therefore, to create a system that allows for a deployment which is tailored to each specific case, the system will in one way or another allow for a simplistic and convenient solution to define and perform an evaluation of any given criteria.

Functions and variables

Another bit of the puzzle could be how the system handles, and if even the language would allow for the use of user defined functions. The format of the evaluation done for each stage of a canary deployment becomes central in figuring out how the nature of the functions will be. Following the concept described in this document, the deployment is to be considered as a process divided into a specified amount of steps, where the evaluation takes place at each point in this series of stages. How the functions are to operate during these stages is something that ought to be considered, as there are possibilities of increasing the value and range of the Manager in how this is handled. As the evaluations are run, both native system methods as well as the possible user defined functions are used to assess the newly introduced environment’s performance, either in comparison to the previous environment, or up against thresholds and levels predefined for the deployment. Should the results and variable states of each stage be persistent and ”globally” accessible however, it is possible that one is opening up for a different and potentially better way of discerning the success of the deployment. Given that the data produced in earlier steps are accessible to the evaluation of the current step, the granularity of the performance indicators can be increased by not only comparing version A with version B, but also how each step has been performing so far, and how the onset of the deployment has affected the service. Having the variable output of each evaluation or increment stored as a time-series or list allows for the developer to tailor criterion in a more detailed manner, and would potentially better accommodate more complex evaluation regimes, where e.g. the variables used are differentiating between stages, or where the increments are not only limited to diverting traffic but includes including new material after a stage is successful.

The idea of persistent variables may prove to be superfluous data, and the functionality proposed might scarcely see usage, but there should be scenarios where this would be of interest, especially where the success of the deployment is of a more critical matter, and a higher resolution of the health indicators is vital. At this stage of development, it is possibly better to leave doors open than to make sure they are shut when developing the system.

4.2 Main Language Features

Having created a potential scaffolding consisting of idioms, ideals and challenges that needs a solution, the first phase of the design process has concluded, and the next phase begins. In this section, a tour of the resulting main language features responding to the needs of the previous section is taken on.

4.2.1 Steps

With the goal defined as a gradual overtaking of traffic from a previous environment to a new, the transitional aspect of a canary deployment has multiple different approaches in ways to be solved. Whether it is a linear increase in traffic with a high granularity, or done through larger leaps controlled by loops or recursive function calls, they both have in common that they regardless of size, are ascending in steps towards a full takeover of traffic directed to the newest version introduced.

In this proposed system, a step is defined as one block of a series, entailing the criteria for progressing towards the desired end state, where specifications of when to progress, and when to abort and revert the process to the starting line. Within a step, the user defines thresholds, comparisons, and Boolean tests which serves as the criterion for the step passing or failing the evaluation which is run at the end of a steps duration

Not all users of the system would have a need to customize the steps, having them alternate between what metrics are to be of importance and plan out a more compound deployment stratagem. Therefore, a default configuration for the deployment steps could be devised and included, allowing for an even more simplified usage of the system. Having the user then only define one or more criteria for success, without further input the system would have default values for all otherwise required fields, such as duration, and amount of steps and their shift in traffic direction.

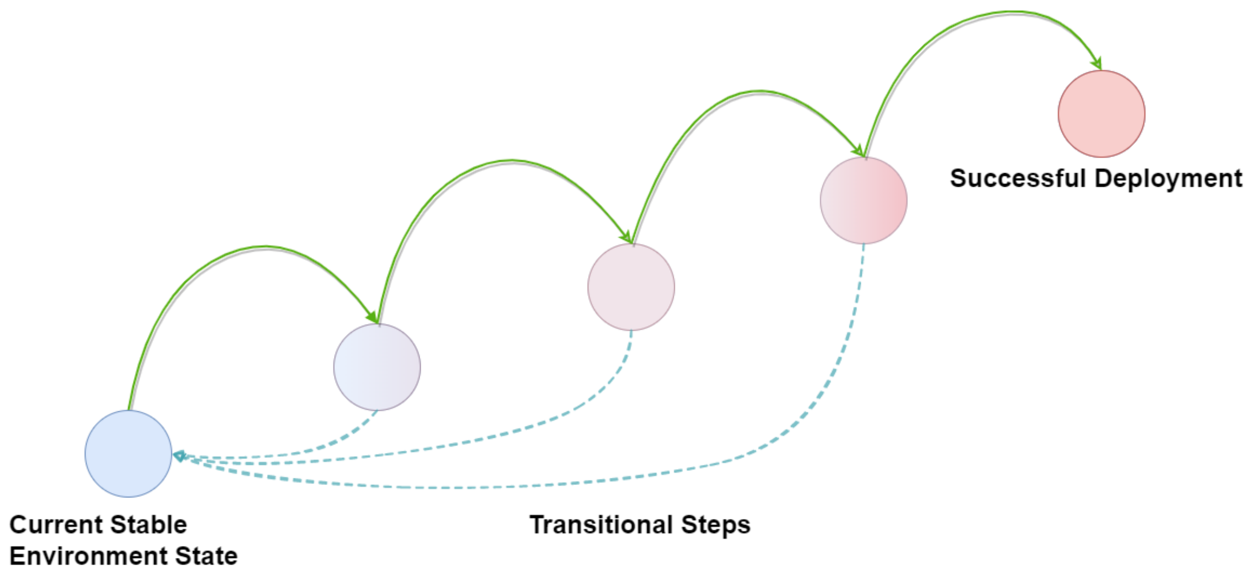


Figure 4.1: The process of the deployment is based on transitioning through steps towards a goal.

Regardless of the content of a step, the process will follow the illustrated sequence as shown in figure 4.1, where through a series of steps, one transitions from a stable environment, to a successful deployment, which in turn resulting in a new, stable environment state. Throughout the process, all steps have the ability to abort the process, and return the the original environment state.

4.2.2 The evaluation

One of the key features of a canary deployment is the ability to retreat from an unsuccessful attempt at introducing an update. To achieve this, an evaluation of the current state of the canary is made, and based on the metrics provided by the canary-environment, the deployment is either terminated, and rolled back, or proceeds to increase the flow of traffic to the canary. This thesis proposes that as this element is of such importance to the process, it is an implicit function not exposed to the user, but in stead run at the conclusion of a steps duration automatically. Maintaining an open ended solution, i.e. aiming for flexibility of use for the system is one of the central thoughts behind this design. And as the diversity of potential use cases for such a system is as wide as there are potential users, attempting to plan for a fixed evaluation plan that will suit every seems improbable, and will most likely result in a design that will seem rigid to more than some. Therefore,

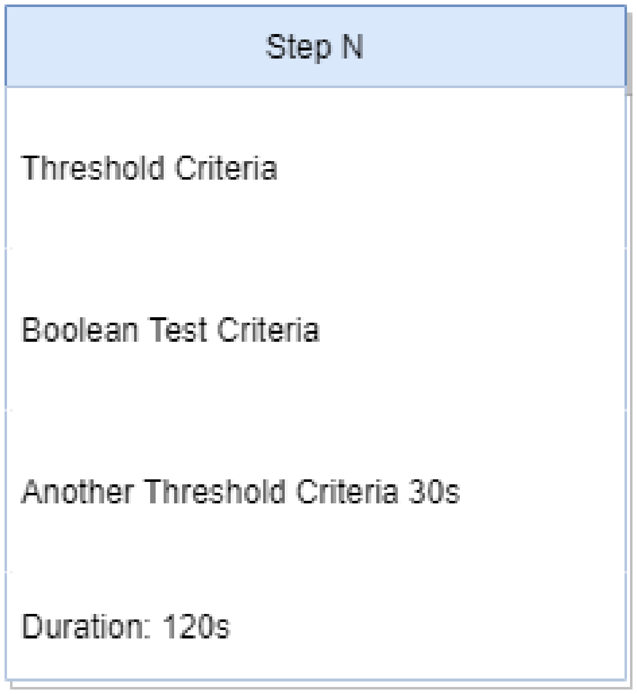


Figure 4.2: An example illustration of an evaluation block

As seen in the snippet above, an abstraction of a single step block can be seen. Within the step, three evaluation criteria is listed, alongside a specification of duration for the step. The duration indicates how long the system should wait after initializing, and in turn progress to the next step, but also subsequently describes at what point in time the evaluation should be run. As the step needs to validate its success before allowing the progression of the canary, it seems natural that a evaluation of its performance is done right before progressing. This selection of timing is also supported by the notion that the step might need time to allow for run-time to accumulate data, and allow for scenarios to arise that does not necessitate an instantaneous occurrence. On the other hand, it would not be a long stretch to assume that certain health indicators are better measured at a more frequent interval than once per step, which in certain scenarios could span across much larger time frames than the duration depicted above. Therefore, as can be seen in the third criteria in the snippet, a simple statement of ”30s” or half a minute is appended to the line, instructing the system that this criterion is to be evaluated independently from the final evaluation, with a frequency of 30 seconds. The evaluation is simplified to the script holding the numeric or Boolean values one wishes to ascertain, when to test it, and how to reach the metrics through the objects of concern. Neither is the actions taken should a test fail or hold the specific value warranting a reaction, but the lack of these details will be covered in a later section.

4.2.3 Inheritance

One of the strengths of operating on a object-oriented based model is one of the paradigms' core concepts, being able to derive the features and abilities of an already defined object unto another, allowing a new class or instance of a entity inherit the properties of a previously defined block of code. This not only lessens the workload of the programmer, but also reduces the number of lines in the program, arguably increasing its aesthetics and readability. The idea behind inheritance is that when a class is produced that will function similarly or almost identically to another class, one can specify that the newfound class will contain all that is defined within the class one designates as its predecessor, which can easily be stated by referencing the parent class in the creation of the new one. This can be handled ”under-the-hood” by the system, by creating inheritance-trees to keep track of the ”genetics” that are to follow in the instance of a new successor.

In most cases, the reason to define a class through inheritance is because one wishes to have a similar behaviour as the parent, but not identical, i.e. that some traits or abilities differ. This is solved by either overwriting the preexisting functions, or by simply adding them to the genetically similar class. In practice, this means that the body of code found in a class based on inheritance will be comprised only of additions or modifications made, resulting in a sleek and slim definition. Where this strategy thrives the most, is where the system will have to handle objects that have the same core mode of operation and are similar, but will need to vary slightly. A potential well suited fit for this, is the concept of the steps in the canary process. As the canary deployment requires the process to be incremental and span across a series of states, which will be described in blocks detailing a step, there will exist a portion of the steps’ behaviour which will unavoidably be identical (e.g. in the manner of executing an evaluation, terminating or proceeding to the next step etc.), as well as features that could be desirably shared. The system, providing the user with mechanisms for propelling movement from one step to another, would therefore only ask of the user to specify what to evaluate for each step.

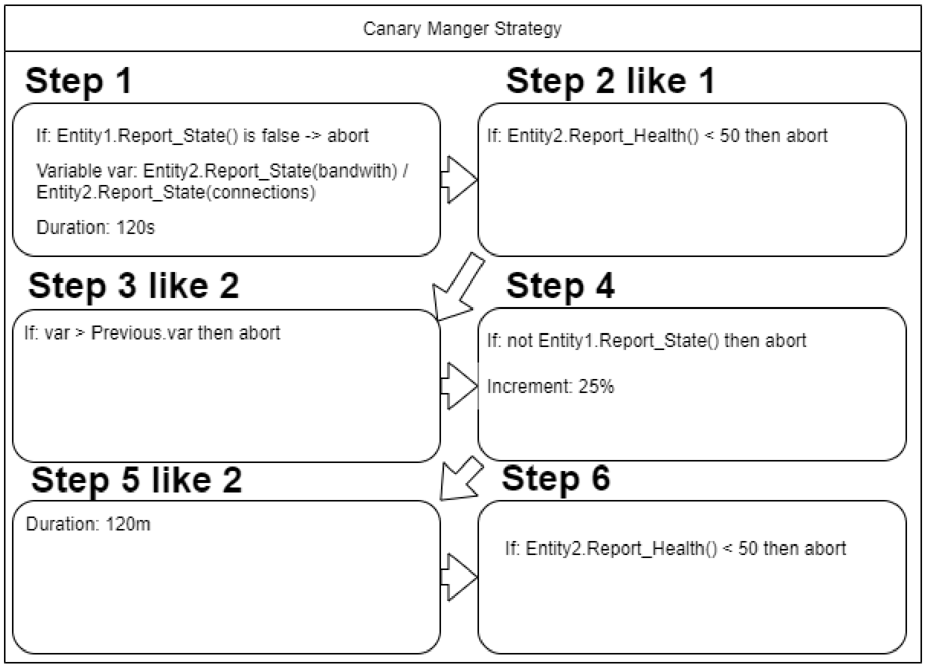

Canary Manger Strategy
Step 1
If: Entity1.Report_State() is false -> abort
Variable var: Entity2.Report_State(bandwith) / Entity2.Report_State(connections)
Duration: 120s
Step 2 like 1
If: Entity2.Report_Health() < 50 then abort
Step 3 like 2
If: var > Previous.var then abort
Step 4
If: not Entity1.Report_State() then abort
Increment: 25%
Step 5 like 2
Duration: 120m
Step 6
If: Entity2.Report_Health() < 50 then abort

Figure 4.3: Illustrating steps employing inheritance

As seen in the diagram above, the initial step defines two criteria which the canary environment ought to stay in bounds of to be considered healthy and successful, both of which will be evaluated after two minutes. In the succeeding step, only one criterion is stated, but as the step is initialized, it is instructed to be a descendant of the preceding step by invoking a command (”like”), and referencing the step to inherit from. Therefore, the second step’s true format would be akin to the first step, both duration wise, and with the same criteria, only with the addition of its own singular function. The third step follows in the same steps as the second, only appending another threshold to the evaluation, before the fourth breaks the lineage.

Sidenote: in figure 4.4’s step 4, a statement about the increment is invoked. In the previous steps, the increment size was omitted, leaving it to default based on step amounts. This change results in the following step size sequence: S1: 16.7% , S2: 16.7% , S3: 16.7% , S4: 25%, S5: 12.5%, S6: 12.5%

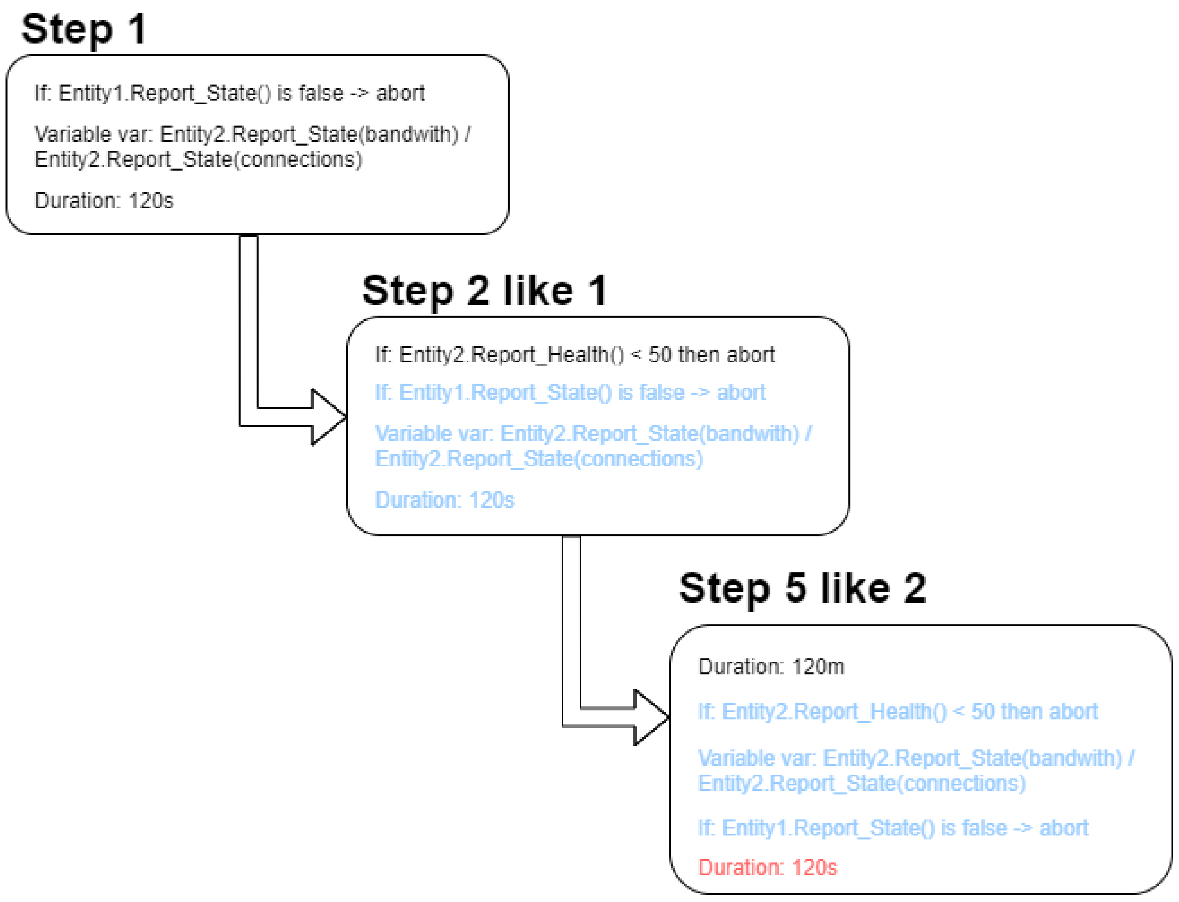

Step 1
If: Entity1.Report_State() is false -> abort
Variable var: Entity2.Report_State(bandwith) / Entity2.Report_State(connections)
Duration: 120s
Step 2 like 1
If: Entity2.Report_Health() < 50 then abort
If: Entity1.Report_State() is false -> abort
Variable var: Entity2.Report_State(bandwith) / Entity2.Report_State(connections)
Duration: 120s
Step 5 like 2
Duration: 120m
If: Entity2.Report_Health() < 50 then abort
Variable var: Entity2.Report_State(bandwith) / Entity2.Report_State(connections)
If: Entity1.Report_State() is false -> abort
Duration: 120s

Figure 4.4: Illustrating the inheritance between steps

In the second figure listed above, contains some of the steps from the previous figure (4.4) with the inheritance line drawn from step 1, through step 2 to step 5 is shown, with the intention of the figure being to illustrate the functionality of the inheritance feature. The commands listed in black font represent actual statements that are written by a user, such the entirety of step 1, where a evaluation failure criteria is set, alongside a variable creation, and a duration specification. In the succeeding step, a single threshold is made, while all the details from the previous step is included, as indicated by the blue colored text. Step 5 behaves in a similar manner as its predecessor, only now overwriting the inherited duration as seen in red, by specifying the duration explicitly.

4.2.4 Scope

As the system is branching towards a object-oriented design, with components being self-contained and steps taking the shape of somewhat independent ”baubles”; existing in separate time-frames, some questions of how to manage variable access surface. With the suggested step structure, multiple steps will interact with the same variables, only they will potentially represent different values as the canary progresses through its states of increased traffic, e.g. network load for step two, will most likely differ from network load in step three, even though they are accessed in an identical manner. What makes this design hurdle even more pressed for a sleek solution is when considering the scenario of a step wanting to compare a current variable value with one of its predecessors, or even the collective average of all steps preceding itself. To rephrase the conundrum; how should one tackle the scope of variables throughout the duration of the deployment? Considering the nature of the metrics produced throughout the deployment, i.e. simple numeric, bit-, or potentially string values, the total size and weight of produced metrics could be insignificant compared to the volume of storage on modern computers. Therefore, accumulating and storing every measurement, even with attached metadata pertaining to which step produced it and when, would arguably not burden the entity hosting the system in a exceedingly noticeable manner. This thesis proposes store these variables in a collective entity, inherently accessible by every step, effectively rendering the issue of access specifiers moot, and allowing the developer to utilize the data with a large degree of freedom. This approach however, places a requirement on the system to provide a method to target the accessible variables with both precision and ease.

In multiple programming languages such as PHP and Python the use of ”this” or ”self” to reference the current class object, is the solution for handling references to the desired scope and context. Though this practice is common and a well known design pattern for many software developers, this thesis operates on the assumption that a reference to any variable will, without added context references, be implicitly intended for the immediate local entity, thus rendering the usage of pseudo-variables such as ”this”, superfluous. But on the other hand, the concept of specifying prepending context to variable references will still serve a purpose, as it makes for a simplified and intuitive method of navigating the ecosystem of available objects. Accessing specific previous steps through dot notation is a simpler procedure to solve, as they are by design named entities with unique titling, allowing the user to easily reference their variables. These design thoughts will be revisited in a later section for advanced features.

4.3 Implementation Specific Features

In the previous section, the core components of the canary manager were described and put in context. Having defined the entities that are assumed necessary, or tackling the arguably unavoidable system needs, the language is now theoretically capable of describing a canary deployment process, and allows for customization tailored to user needs. However, before employing the Manager to a test, more work is in need of presentation. This section will discuss the features which are less probable to be considered an intuitive design choice, but are central to achieving the desired simplicity which is listed as one of the goals of the project.

4.3.1 A Library of Drivers

One of the conceivable challenges that arises when theorizing about introducing OOP into the realm of deployments, is the sheer amount of work that it would require. As the process of deploying involves a potentially vast selection of different entities, with imaginably an even larger number of components, the system would be teeming with different object definitions. In a declarative world, one does not need to describe the entirety of the entities comprising the environment, but can make due with describing how to reach the designated component, and the desired interactions with them. With an object-oriented approach on the other hand, one wishes to describe the entities with all their functionalities and characteristics to fully encapsulate the component and its responsibilities within the system. Should one however, take the step of producing object descriptions for all the entities that a deployment entails, some advantages and possibilities arise. With an object representation of a component, it becomes an entity which the user can reach in a highly simplified manner, which in turn makes the process of creating and managing criteria for evaluation less demanding. Instead of having to script predefined interactions for each entity within the environment, one could now rely on predefined functionalities residing within the objectified components.

To tackle the potentially daunting task of mapping all potential tools and services that could be expected in a software environment, two propositions are made.

The canary manager provides a standardized format for a driver

Providing a standard for object-notation of components would alleviate some of the work required by those employing the manager. A template, of which is already understood by the system, could simply provide the end user with segments where the necessary details are input, such as filling out the directory which the entity resides in, detailing the possible interactions the tool offers, and necessities such as authentication needs or configurations. In short, this driver would be a mapping of the capabilities of any given component, in a manner that allows the manager to invoke its functionalities or alter its features. Providing such a standardized driver would likely greatly lower the threshold for users to approach the use of the service, again keeping true to the principle of simplicity. But despite the reduced complexity of driver-creation one could provide, the process would still prove to be non-trivial in manners of time consumption. A large environment could still be housing a tremendous amount of components, rendering the work of driver creation not only drawn out, but also repetitive. This situation led to the second proposition.

A community centered around the creating and sharing of said drivers

Given a widespread use of the canary manager, it would not take long before most of the commonly known tools had seen a driver created for them. Though they might vary slightly on details such as custom configuration and directory placement, they would carry the same capabilities across the board. While some times it could prove necessary to configure such a driver from the scratch, more often than not one would only be reinventing the wheel while doing so, as previous users of the manager will have had to go through the process of erecting the driver files for their own system. Therefore, to combat this, as the proposition suggests, an open source community surrounding the canary managers' drivers is envisioned.

The community-shared content will be comprised of completed drivers for different components, created as the need occurs. And as they are created, the driver template ensures the now objectified components maintained to some extent a uniform fashion of interaction. As additions are made, the community hub will serve as a library of drivers for a vast variety of components, pertaining to both different tools, as well as different versions. This catalogue of drivers would enable future users of the manager to browse for their selection of components and download them for their own usage.

Should the driver library reach a sufficient size, where one no longer need to check for existing drivers, but rather can expect their existence an even more simplified method could be employed. A situation like this would allow for a functional integrated pull-based command, affording the user to only state the name and version of the components they are utilizing, and the manager will be able to pull the required drivers to their codebase.

Similar concepts of shared ”recipe libraries” exists, and even thrives. One example with quite similar qualities, would be the Docker Hub, where, according to their catalogue, can find over nine million container images contributed to their community (Docker Hub, 2022). Of these, 174 are classified as official images, meaning they are created by the same people developing the service they are representing. Should the canary manager gain sufficient traction, it is not unthinkable to see similar behaviour from software development organizations as seen in instances such as the Docker hub, where they would provide an official driver for their tool or service, and maintain it with upgrades and patches as their own service develops.

4.3.2 Following the Codebase

Though one could argue a Canary deployment is more about damage control and safety in deployment rather than continuity, it exists within a sphere that is heavily influenced by the principles of continuity, both in integration, and in deployment. Therefore, as the canary manager would likely be set up in environments where continuity is integral to the values of those who offer the service, it seems important assist in reaching this value. As the design went on, a question surfaced as to how the canary manager is set up and integrated. This brought light to the certainty that should the canary manager require more than minimal effort of manual configuration each time it is activated, it could be obstructing the work towards continuity, rather than encouraging it.

To work around this awkward position, an implementation feature was conceived. Upon first setup of the canary manager, a ”scaffolding” must indeed be set up, where the necessary drivers are specified, configuration of access and the detailing of the deployment strategy. But as this is set up, the files pertaining to the canary manager is stored within the service codebase, included as one would with any other configuration’s files. As newer versions are then created in parallel replica environments, the deployment strategy, driver integrations and other necessities are already present, and would only require small adjustments to accommodate eventual alterations to the system structure that requires representation within the canary manager.

An illustration was created to aid in relaying the concept, which can be seen below.

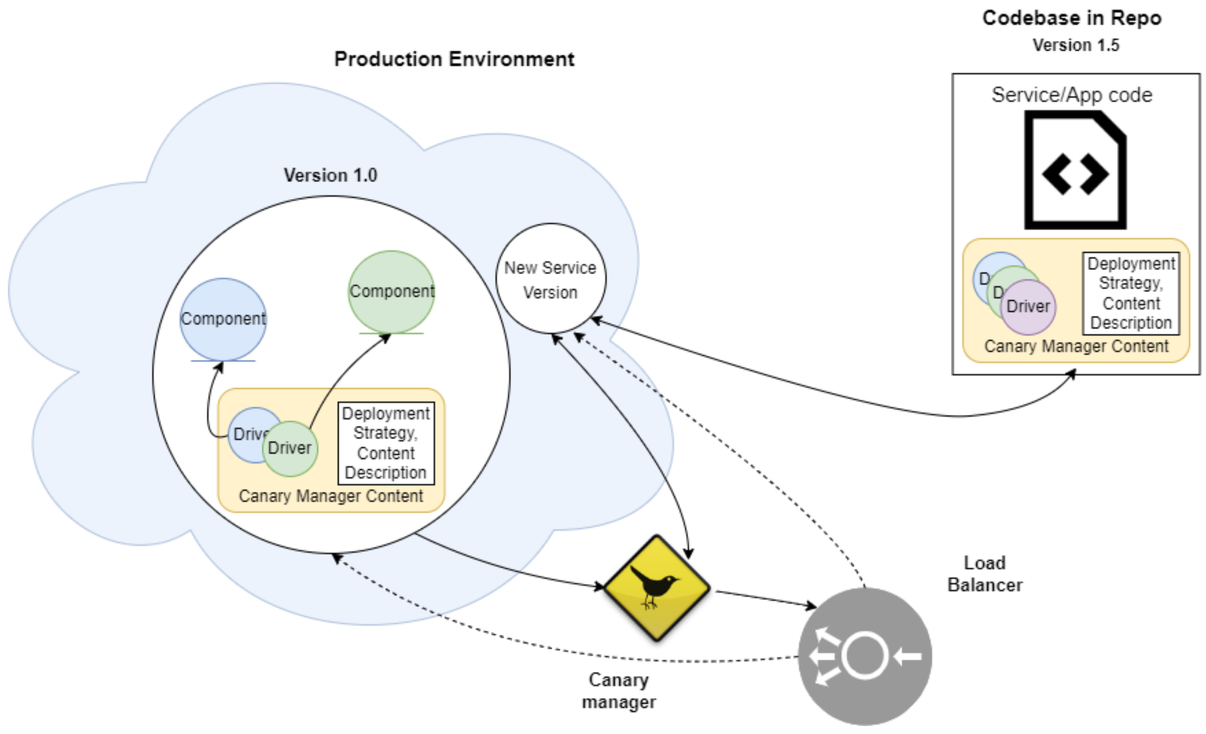


Figure 4.5: The Canary Manager, with access to a newly spawned service pulling content from a repository

In this miniature universe, a production environment is running the current version of a service which is made available through the control of a load balancer. The service has either previously been deployed with the canary manager, or has been fully configured for use, as it contains drivers and has a deployment strategy contained alongside its codebase. On the right hand side, a commit has been made to the connected repository, expanding the service and as seen within its canary manager content, an additional driver as well. A new host, such as a container is spun up in the image of the new version detailed in the repository's edition of the code. Once the new host has proceeded through the pipelines' stages, presumably being exposed to rigorous testing, and is ready for introduction to real traffic, the Canary Manager takes control. A step sequence detailed by the manager-content included in the pull from the repository allows the manager to automatically decide on how to proceed, and is conversing with the load-balancer to introduce traffic to the newer version as instructed ”on the package”. As the driver configuration has been completed in a previous version, their connection to the system is already functioning and the manager can begin measuring data both from the previous version, as well as the new service being introduced as a canary. The Manager follows the strategy as far as possible, and reports the outcome to whichever channel or medium is instructed.

A comment to this abstration: In this illustration, the pipeline leading up to the creation of the newly spawned service is omitted, as there are no singular way to achieve this. The canary manager however, is intended to be independent of surrounding entities and therefore, the inclusion of a specific release pipeline becomes moot.

4.4 Advanced Concepts

In this section, features or ideas that are not necessarily a part of the core of the language, as well as features that are not yet fully conceived are presented. These can be counted as either suggestions, or as elements that still require additional work, but are intended as a part of the Canary Manager package.

4.4.1 Advanced Step Order

As mentioned some times throughout this thesis, allowing for the option of higher control or granularity in the use of the system may be essential for the language, even if scenarios employing it cannot be envisioned at the moment of writing. One feature which lands within this description is the capability of advanced step order.

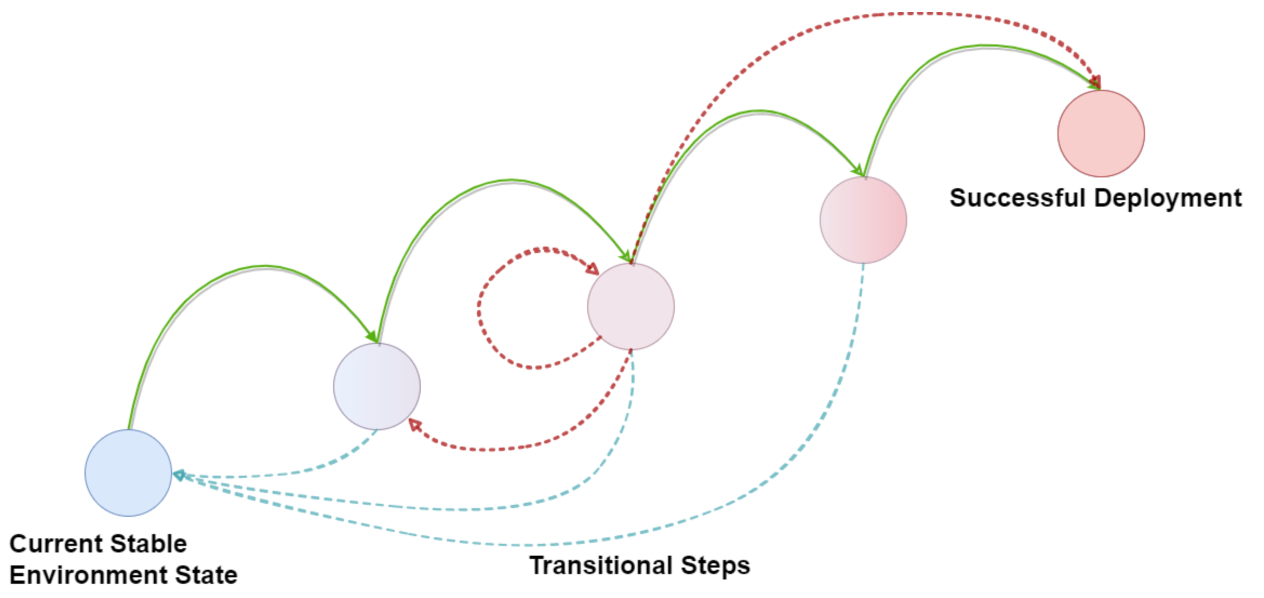


Figure 4.6: A step sequence with optional routes

With the figure above (4.6), the same pattern of progression as presented in the section detailing steps is shown, except for three additional routes being postulated, as seen by the (red) dotted lines originating from the center transitional step. A question of ”are there scenarios where one would either return to a previous step, repeat the current, or The first new addition was theorized by considering whether there would ever be a scenario where one would wish to retreat a step or more, as opposed to terminating the deployment. Similarly, the option of re-doing a step, initiating the sequence of measurements and evaluation one more time. Both these optional routes could prove to be valuable, should the manager have sufficient ability to detect and more importantly correct the source of the fault resulting in a repeat/retreat. In simple situations, the step might only had needed more time to pass as successful, or a retry of function-calls would sort out any mishaps that led the system to abort, and a retracing of steps, or even just a rerun of the occurring step might be a valuable and time-saving feature. However, it is possible that the level of advanced problem solution capabilities required of a manager capable of performing ”self-repairs” may be too high to be feasible.

The third and last optional sequence introduces the ability to skip ahead, as seen in the illustration where the third step has a route directly to the final stage. It is not unthinkable that should a deployment strategy containing an array of multiple steps have been experiencing green lights throughout the entire process, it could be possible to proceed to speeding up the process by either skipping steps, increasing the increments to shorten the length, or concluding an early success and jumping to the last step of the entire strategy. That being said, this could prove to be a complex feature to program, with potential low yield for then end user, resulting in a lot of work for little value. This is potentially a feature that might need to investigated further to decide whether it is worth consideration or not

4.4.2 Immediate Events

A feature which has not been given too much thought, but may prove to be a necessity for the Canary Manager, is the ability to respond to immediate events. As per the proposed design, evaluations and subsequently the metrics used are being tested either at the end of a step, or at their specified interval. But there might be occurrences during a deployment which would warrant the immediate abort of the process, which by this design, would only happen should a metric be encapsulating this scenario, and at the pace of which the metric is set to be evaluated. One could solve this by including instructions to measure such metrics at a much more frequent interval in a location outside the scope of a step, having the manager testing the conditional parallel to the evaluations of the steps. This solution however, presents additional areas of the Canary Manager's content which would need configuration, a notion that does not align too well with the ideal of simplicity. Regardless of the angle of implementation, this feature may be one that will be of the more protruding necessities given further development of the manager.

4.4.3 Global and Local Scope

Though some of the scope features has been discussed in a previous section, there are some quirks and aspects that are still in need of further exploration.

In a scenario where one should wish to have all the steps perform an evaluation of the same metric, one could do this through inheritance, configuring the initial step to contain the desired measurement and have all subsequent steps inherit this trait. However, this puts the developer in an awkward position of not only having to ensure all steps inherit from the initial step, but also potentially limiting the contents they are able to include in the block for the first step. To provide a solution to situations of this sort, an addition is made to the instruction set; a global variable block, and a global object.

Before detailing the progression schedule and strategy of the canary by creating steps, one can choose to initialize the deployment by declaring variables that are to be tested or measured at every step, regardless of inheritance. This also provides the developer with a simplified method of creating arrays or collections of metrics which can be accessed by any step throughout the process of deploying.

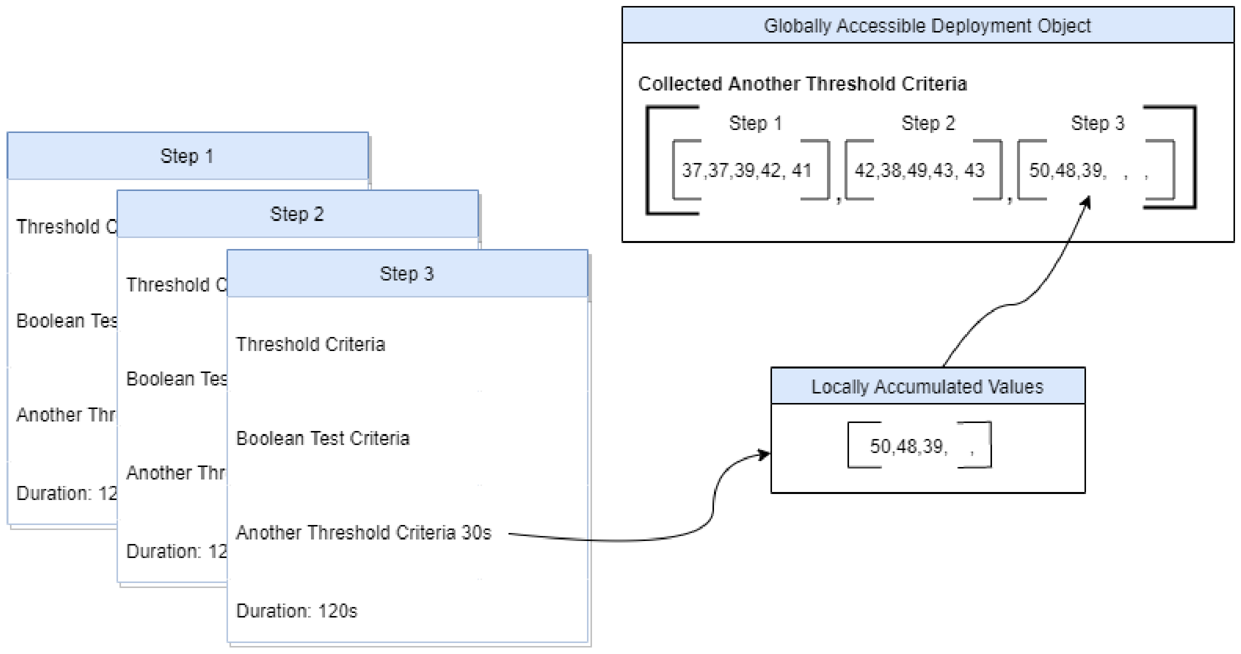
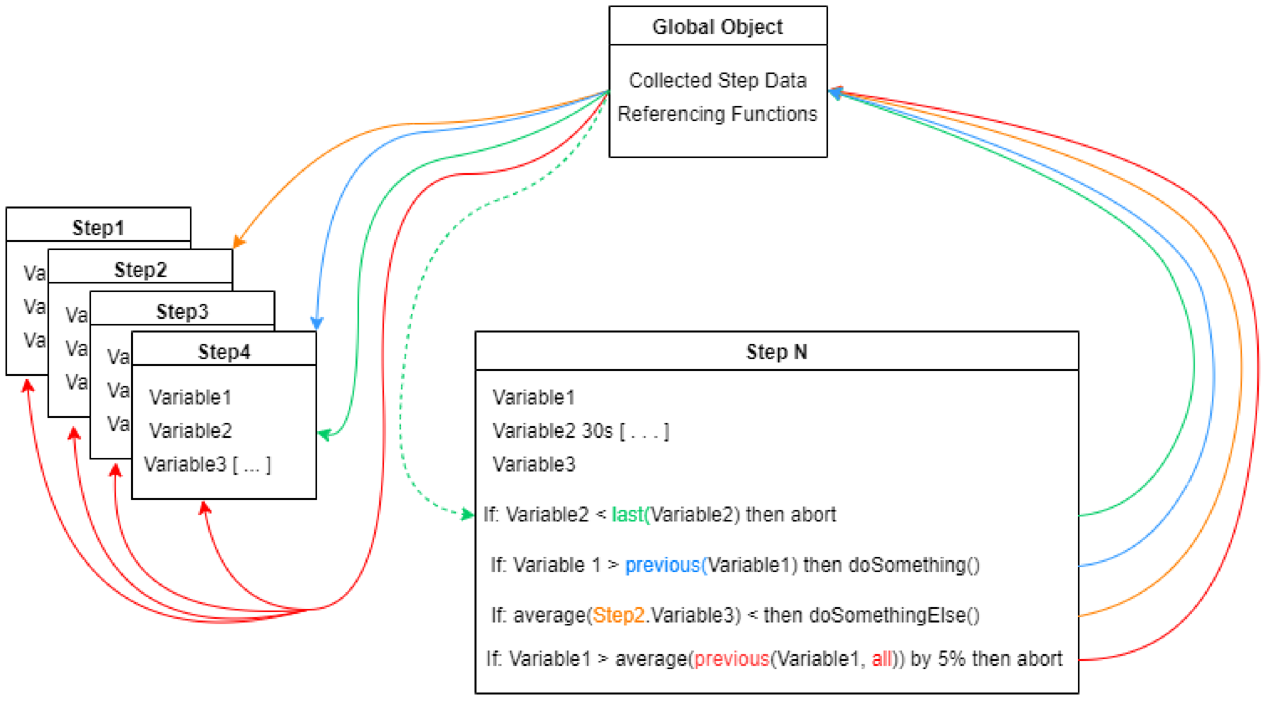


Figure 4.7: Variables' accessibility throughout canary process

In the illustration above, a sequence of steps is depicted, and the envisioned concept of global variable storage is portrayed. The figure shows three identical steps, with four descriptive statements. In this example, the criteria used to visualize the functionality is of a sort that requires a more frequent measurement, being evaluated at every half minute. In this design, these variable values are collected into a series, representative for each step. Either as the step is progressing, or as the step is finished, the resulting values, such as the threshold series in the figure, are passed on to another object. Upon reaching the evaluation function at the end of the duration of a step, a method call to the global object would instruct it to complete the current step as a finished object, initializing a constructor for the creation of the next step.

What this thesis proposes, is that regardless of the variable type (singular, series, numeric etc) and their origin, these values should be automatically stored semi-persistently, creating a consolidated collection of all data produced throughout the deployment process. This object, is a globally accessible structure which contains the accumulated evaluation data from all the steps throughout the deployment runtime. The global object allows for an arguably simplified method of interacting with data from the preceding steps, a functionality illustrated in the figure below



Global Object
Collected Step Data
Referencing Functions
Step1
Step2
Step3
Step4
Variable1
Variable2
Variable3 [...]
Step N
Variable1
Variable2 30s [. . .]
Variable3
If: Variable2 < last(Variable2) then abort
If: Variable 1 > previous(Variable1) then doSomething()
If: average(Step2.Variable3) < then doSomethingElse()
If: Variable1 > average(previous(Variable1, all)) by 5% then abort

Figure 4.8: Accessing deployment data through the global object

In the figure above (4.8), the imagined workings of this global object is illustrated. As can be seen, a series of steps have been performed, and the system is set to proceed with the current step, ”Step N”. ”N” contains or generates the three same variables as its predecessors, but now also contain a series of conditionals which are created to demonstrate the use of the global object, and it’s ability to pinpoints data references throughout the deployments timeline. These references are envisioned as function calls, which are innately tied to the global object, and follows the syntactic structure shown below.

```
targetFunction( target variable, optional modificator).
```

Back in figure 4.8, the first conditional (green), is making a comparison to the last instance of Variable 2, which in this situation, might either be the last recorded entry in a series found in the previous step. Should however, the step have experienced a duration of over 30 seconds, the most recent entry of Variable 2 will be found within the current step. As the global object has access to all entries, it simply jumps to the last entry of the last object detailing Variable 2. Similarly with the second conditional (blue), a reference to the second to last step is done by accessing the data added from the step prior to the current.

With the third (orange) step, a direct reference to a specific step is made. A scenario where this might occur is where the specified step contains metrics that is omitted from the others, and one wishes to only intermittently test for these conditionals. The reference is achieved by providing the name of the step one wishes to access, and as the global object maintains an overview of the object structure of the steps, is readily available through dot notation. In the last step (red), some additional features are shown.

Firstly, the reference is now pointed towards all occurences of Variable 1 in the preceding steps, by using the modificator for ”all” previous entries, excluding the current as use of the ”previous” target function sets the scope. Averaging the sums, a function believed to be both necessary, and expected in the canary manager, will provide a sum which is compared to the current value of Variable 1. However, to alleviate the user from dreadful basic maths, the use of another function-operator; ”by” instructs the system to calculate the value of Variable 1 increased by 5%, before deciding whether the new sum is within bounds of the preceding average.

Such operations as This < That by 5% will require the system to relate the differential calculation to the comparison operator, making it a bit more complex implementation-wise that an ordinary arithmetic variable. That being said, this format of the syntax provides a high degree of readability, allowing the user to format thresholds with leeway by simply stating ”if This is Greater than That by a Percentage of ...”. These forms of operators are concurrent with the statements made about the need and potential value of more advanced operators made in the initial phase of the design, and their discovery is most likely to be followed by more should the design process continue.

Chapter 5

Analysis

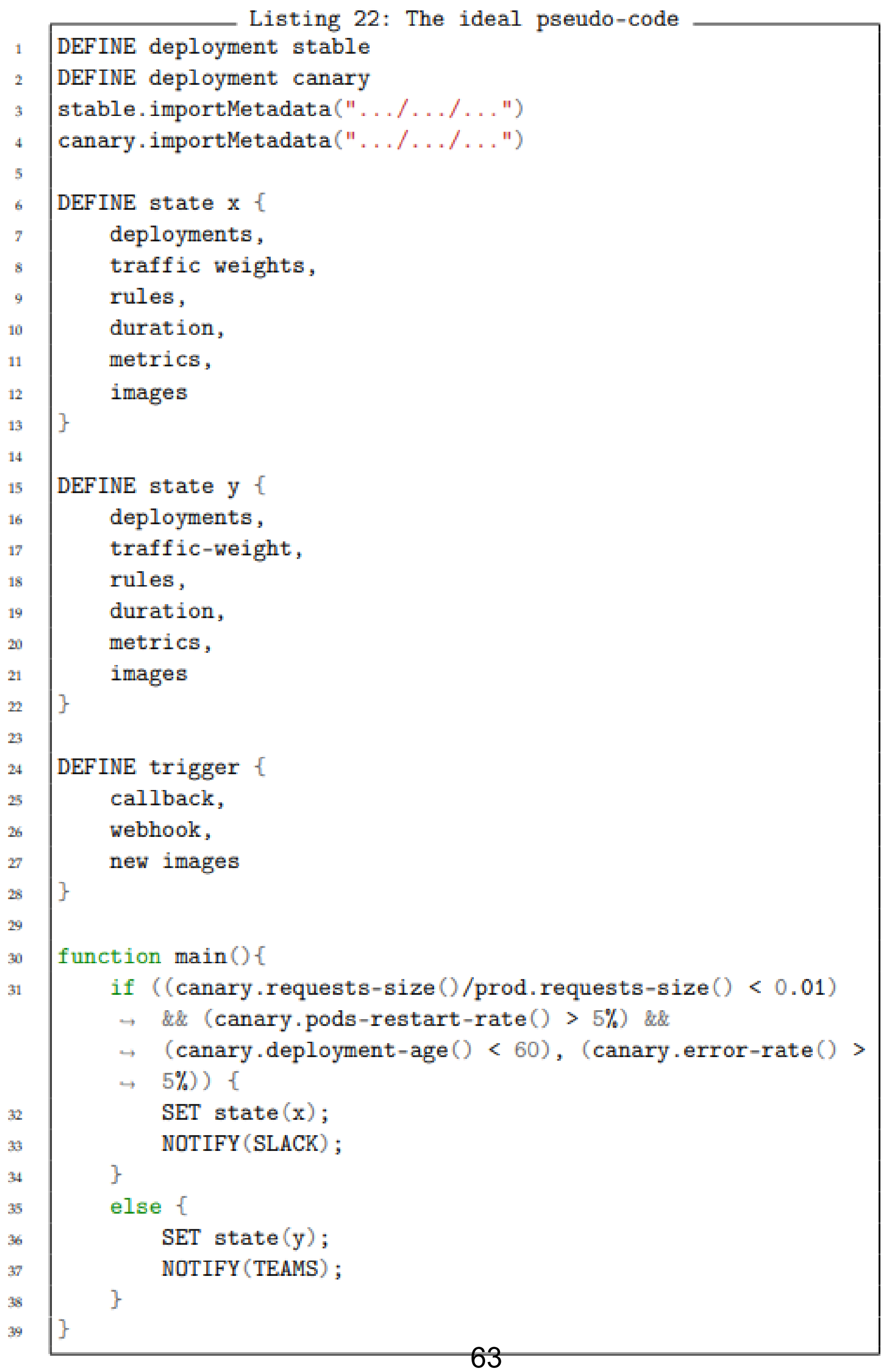
To ascertain the resulting capabilities of the design, and to discern the ability of the system to respond to the problem statements,a dissection of the proposed Canary Manager is done by subjecting it to a two-part analysis with differing approaches. In the first method of analysis, an actual predefined situation with canary deployment will be used, bringing a bit of realism into this otherwise theoretical and abstract realm which the language resides within. Upon completing the first analysis, a resulting file containing a solution to its given scenario will serve as the material for the second part of the analysis. Keeping parts of the original deployment scenario, but making alterations to the specifications of the initial desired evaluation properties, and the corresponding alterations to the manager file will provide insight into the system's ability to respond to the problems presented. Along the way, features that appear but have not been previously discussed may be highlighted.

5.1 Wielding the language

In her fourth and final design of the master thesis, Çeliku presents a situation containing an arguably complex example situation. The case she provides is a series of conditional statements, all dependent on each other reaching a specific state before the canary is decidedly terminated. The example situation is described as such:

”If canary’s requests payload increases with more than 1% compared to the stable version and it is longer than 1 hour since the canary has started, and if there has been a 5% increase of HTTP error messages, and Kubernetes has told us that several of the containers in the canary deployment have been restarted, then in that case and only in that case, we shut down the canary!” (Çeliku, 2021, p. 107).

With this problem space defined, she presents a script of pseudo-code, which is shown in the following figure, found on the next page.



Listing 22: The ideal pseudo-code

```
DEFINE deployment stable
DEFINE deployment canary
stable.importMetadata("../../...")
canary.importMetadata("../../...")

DEFINE state x {
    deployments,
    traffic weights,
    rules,
    duration,
    metrics,
    images
}

DEFINE state y {
    deployments,
    traffic-weight,
    rules,
    duration,
    metrics,
    images
}

DEFINE trigger {
    callback,
    webhook,
    new images
}

function main(){
    if ((canary.requests-size()/prod.requests-size() < 0.01)
     ↪  && (canary.pods-restart-rate() > 5%) &&
     ↪  (canary.deployment-age() < 60), (canary.error-rate() >
     ↪  5%)) {
        SET state(x);
        NOTIFY(SLACK);
    }
    else {
        SET state(y);
        NOTIFY(TEAMS);
    }
}
```

Figure 5.1: Proposed pseudo code from the publication ”Continuity-as-Code”.(Çeliku, 2021, p. 111-112)

The snippet shown above, spanning over 39 lines, is arguably not a very lengthy instruction set for the otherwise potentially complex task of canary deployment. In a language that bears resemblances to commonly known languages such as PHP, the script and its intended functions can be broken down into three sections.

The first revolves around preparation of the workspace, where namespaces for the environments concerned are set, and their data is loaded into the system for usage throughout the deployment. In the second section, a series of definitions is made, describing two different states, the first which will be set upon failure, the second containing the state details which are to be implemented should the canary succeed. This demonstration also includes a trigger, which is proposed as a method of influencing the states as seen fit, by including callbacks, webhooks, new images and more (Çeliku, 2021, p. 110). Lastly, a function which drives the deployment, by testing conditionals and invoking corresponding state on the environment. One of the key values this approach provides is a freedom in terms of accessing and making use of data from either environment, as well as an almost unlimited possibilities of concocting conditionals. On the other hand, the unspecified variables within the definitions seem to be indicative of more work than what is displayed in the pseudo-code.

With the predecessor presented, the Canary Manager's language is employed to the same situation with pseudo code based on the resulting design of the previous chapters. As seen in the manager example below, some new, previously undisclosed features are shown in the pseudo-code. Firstly, the existence of measurements intrinsic to the system, such as keeping track of the run-time of the process.

```
Step 1{
    increment: 10%
    duration: 15m
    if: [
        global.runtime > 60m
        kubernetes.pods-restart-rate > 0.05
        apache.requests-size > old.apache.request-size by 1%
        apache.http-errors > old.apache.http-errors by 5%

    ] then abort
}
```

The first, and only step is defined. Here, three statements are specified, with the first specifying the increment size of the step, i.e. how much traffic weight is to be increased or introduced to the canary. Interpreting from the pseudo-code provided by Çeliku, the proposed configuration iterates over a 10 step cycle, increasing the weight distributed to the canary by 10% for each step. As the step is the only one, the declaration of a 10% weight increase translates to a ten step process.

Following the description of the traffic handling, is the duration of each step being set to 15 minutes each. Already in the first two statements, one is looking at alleviating the user from a load of redundant repetitive work by the system interpreting the weight increase ratio, and translating it to the amount of steps needed, carrying the to-be defined evaluation criteria with it to the resulting replicas of the step block, without composing a recursive loop.

Following the block's temporal and ”spatial” frames being set, the issue of evaluating for failure is tackled. Whereas in the original script by Çeliku uses a series of logical AND operators to define their co-dependency in the matter of the result, the manager employs a more short-handed approach. By bracketing the statements of the preceding IF-conditional, the manager interprets the series of individual statements as a collective value. This method of replacing the need for creating a chain of AND-statements reduces the complexity of describing the scenario leading to a system abort through simple one-liners of code. Though this is only removing the need for ”&&” or equivalent in the case, it arguably creates a much higher degree of readability, which in turn would make it easier to write as well. It is assumed that such conjunctional conditionals will be a common occurrence in the use of the manager, as complex environment states are comprised of components dependent or co-dependent on each other, rendering this otherwise small change a potential high value functionality of the manager. Keeping in mind, that in the same manner of behaviour as logical AND operators function in many languages, the first untrue statement encountered cancels the collective test, and proceeds to the next instruction, saving time and machine-resources in the process. Within the conditional, the step tests for the runtime of the entire deployment process to be larger than an hour by referencing an innately kept variable of the system. Then, in likening to the original code of Çeliku, tests pod restart rate, request size- and http-error growth compared to the old version, referencing it through the use of ”old”, to access it through the global object.

As the state of the deployment is intrinsic to the manager; being incorporated in the code-base, the need for elongated state definitions and prefacing variable or metric initialization is wiped away. As the drivers corresponding to their entities are present within each of the environments, they are readily called upon when creating the conditions for success. In comparison, this would mean that the end of the system the user is exposed to, is comparable to only the main function of the pseudocode in figure 5.1. In other words, the user is alleviated from a lot of preparation work, and is taken almost directly to describe the behaviour of the canary.

5.2 Demonstrating Usage

In this section, the proposed system is demonstrated through making alterations to the original problem statement used in previous presentation. The goal is to present the possible versatility, flexibility and ease of use of the canary manager, by illustrating the necessary alterations to the code following a varied problem statement.

5.2.1 Circumventing a potential abort

Assuming the developer is aware of what might cause the system to reach the state which might cause an abort, but also has a potential workaround, they might wish to have try giving the system a ”helpful smack” before deciding to terminate the attempt at deployment. In this envisioned scenario, something tied to the deployment of the Kubernetes pods is known to cause issues with rolling out updates.

```
Step 1{
    increment: 10%
    duration: 15m
    if: [
        global.runtime > 60m
        kubernetes.pods-restart-rate > 0.05
        apache.requests-size > old.apache.request-size by 1%
        apache.http-errors > old.apache.http-errors by 5%
    ] then [
        retryorabort
        kubernetes.rollout-restart-deployment()
        wait: 10m
    ]
}
```

With the example above, the instructions remain the same, until where the script previously would invoke the abort command. Now, the system is given the instruction to retry or abort, a command that notes its position in the instruction set, and upon reaching the final evaluation of the step, sets the system to redo the current step. Should the manager reach the marker set by the previous encounter with the ”retryorabort” command, the system terminates, avoiding perpetual loops. However, note that following the instruction to retry, the system calls upon Kubernetes to do a restart of its pods, and waiting 10 minutes before proceeding to attempt the step again.

In the section pertaining advanced concepts, the question is asked whether one should consider the need for the ability to iterate over the same step again. As shown with the example above, not only was it needed, but also quite simple to envision a implementation of it as well.

5.2.2 Making it strict

In the original statement, a series of conditionals would all have to coincide for the Manager to deem the Canary a failure. In this version, should either of the situations occur, a rollback is warranted.

```
Step 1{
    increment: 10%
    duration: 15m
    if any: [
        global.runtime > 60m
        kubernetes.pods-restart-rate > 0.05
        apache.requests-size > old.apache.request-size by 1%
        apache.http-errors > old.apache.http-errors by 5%
    ]
}
```

Again, a simple syntactic entity is altering the course of the entire evaluation, by appending ”any” to the leading conditional statement, the encapsulated previously codependent statements have become separate, as if they would be coupled by a logical ”OR” operator.

5.2.3 Introducing tests as the Canary progresses

In the next alteration of the scenario, we envision that the developer wishes to allow the newly initialized environment to run for a bit, before starting the tests, and gradually introducing them as the strategy reaches its conclusion.

```
Step 1{
    increment: 10%
    duration: 15m
    if: global.runtime < 60m then conclude
}
Step 2 like 1 {
    if: kubernetes.pods-restart-rate > 0.05 then abort
}
Step 3 like 2 {
    if: traffic > 50% then if:
        apache.requests-size > old.apache.request-size by 1% then abort
    if: traffic < 80% then repeat
}
Step 4 like 3 {
    if: apache.http-errors > old.apache.http-errors by 5% then abort
}
```

As demonstrated by the above example, the language provides multiple methods of proceeding through the course of a deployment. With this final example, the criteria used to measure success are accumulated throughout the run by means of inheritance, each step containing their own specifications alongside what their predecessor measured. This example mainly illustrate differing ways one can instruct the system to progress, and/or maintain its position on a step instead of proceeding. Three new keywords appear in this example, where ”conclude” is used to inform the step to proceed to the next step should the evaluation of the runtime not meet the specified value. A second, referencing ”traffic” is intended to ascertain the current rate of traffic to the canary, which is used to decide which conditionals to use, or what actions to take. The third, ”repeat”, instructs the step to redo itself until the criteria specified is met, which in this case is a traffic distribution above 80% to the Canary.

5.2.4 Summary

Through the use of two different methods, some of the intended structure of the Canary Manager’s language has been demonstrated. By employing the language on the case presented by Çeliku, it has proven to be able to drastically cut down on code length and structural complexity of instructions. By altering the case description, some alternative usage of the syntax could be shown, and with them, some of the core principles and entities discussed in previous sections as well.

Chapter 6

Discussion

With the process of creation, and subsequent analysis and demonstration being completed, a section for afterthought is presented. Here, each paragraph contains a comment to a specific section of the work that has been done, or posing a question that might not have been answered.

A comment to the perceived usability of Canary deployment

As the discussed presentation of the background chapter comes to an end, the impression left of canaries as a concept does not leave a pleasant aftertaste. With only a few actual features presented as the reward, the required effort, arguably being depicted as a struggle, is overshadowing the value of the strategy he is peddling. This is in stark contrast to the beliefs held by this thesis in regards to what a canary deployment could be capable of. On the other hand, our viewpoints align when discussing the availability of tools and systems capable of realizing the capabilities latent within canaries. McKenna mentions Spinnaker configured with the correct plugins to the be the closest we have to an ”Off-the-shelf” solution, but further stating that you will still be required to produce and maintain manual code (McKenna, 2019). Though extensive knowledge of Spinnaker and its abilities is not something I retain, based on the information encountered throughout the course of this thesis, if it is the most viable option to Canary automation, then the industry is ripe for a system such as the one presented with this project.

Should the process of integrating a canary approach into the workflow of an organization prove to actually require the amount of time and effort postulated by McKenna, one can argue that there are seemingly more effective and rewarding projects to invest one's resources in (McKenna, 2019). That being said, should one increase the yield from the investment, the pitching of the necessary change becomes easier, and subsequently the acquisition of buy-in from the higher levels in an organization, which McKenna states as a must-have. This increase of value is what the canary manager might provide. The proposed design does not only allow for a simplified method of creating canary strategies, but also shows promise of evolving the concept into something more. Yes, there is great merit in the early warning functionality a traditional canary provides, but as this system demonstrates, if one already have access to all the entities comprising the environment, making use of them can enable more than just a metric health-check. The possibilities introduced by having the manager capable of interacting with its fellow entities, could allow the manager to aptly scale the service according to the well-being of the system, alter the states of components as required, or better attempt to solve the obstacles preventing deployment on its own.

Where does the Canary Manager exist?

As the system is presented, the details regarding a underlying design capable of handling the desired architecture is omitted. As this thesis centers around the development of the conceptual framework for the language, the design process has proceeded on the assumption that the inner workings of necessities such as a compiler for the code, the manager's technical integration in the development pipeline and such became trivial. That being said, should one investigate the feasibility of implementing the features of the manager, one would arguably not be able to avoid discussions such as this, and it might reveal flaws, incompatibilities and technical demands that would lead to having to alter or even axe parts of the proposed features of the manager.

Object-orientation as the weapon of choice

In this thesis, it is proposed to implement an object-oriented approach to solve several challenges the language faces, such as repetition of code and for interaction with entities within an environment. The paradigm is selected and presented as a good contender due to its ability to decouple components, and for its method of abstracting these components into solvable problems. But though the orientation is attributed with a level of abstraction, there are other paradigms that may provide an even better fit for the canary manager's goal. It may be due to the current state of the curriculum for students in information sciences programs at universities, that an inclination towards object oriented languages is present. At least from my own experience, with peers and colleagues of relatively similar age, one can find that they are versed in languages such as java, python, PHP and javascript, all of which are to some degree object-oriented. But upon investigating OOP, while uncovering that it is counted as one of the most widespread paradigms, articles and discussions suggesting it also one of the more disliked design patterns were not uncommon. Should one proceed with taking an approach that is universally disliked, it might have grim consequences for the reception and success for the system.

The Canary and the cloud

With her models, Çeliku positions her work in the cloud-based region of the field, which is understandable due to the prominence of cloud architectures today, as mentioned in the background chapter. However, though the statistics presented in this section regarding cloud usage is high, it might not reflect reality accurately. Though many employ cloud services in one way or another, virtualization technology and the use of off-site computation as the main accommodation for a service might not be as common as the statistics indicate. That is why it could prove to be even more versatile and inclusive to create the system based on a higher level of independence, and allow for usage regardless of the underlying technology. As Cisco states in their report, the numbers insinuate an ever-increasing use of cloud, but keeping the doors open for alternative usage might be a better solution, especially considering the possible reliance on open-source contributions.

A thought on the method of this thesis

The projects' exploratory and open-ended method gave the design phase a lot of freedom, which can allow for a creativity which might otherwise be stifled by methods with more rigid structures. However, without cemented target milestones adding a sense of pressure, it is possible that the design process was snubbed of potential additional progress, as it would have been force to produce results based on the goals set in advance. In addition, with the project structure used, the investment of working hours became cluttered around the different parts of the design at an uneven pace. Had the work followed a more structured course, and rather than revisiting a feature freely, had been done in an iterative process, it would have simplified documenting the work put in into the various facets of the resulting design and yielded a better overview. It has now become my experience that a project of this sort, being experimental and investigatory, would have been simper to not only document, but rewarding oneself with a sense of progression, should it have had at least some loosely set sub-goals to navigate towards.

The yield of the analysis

It was through the employment of the model on a case that a lot of needs were uncovered, and the language was introduced to a series of new operators and functions which would prove to solve the situations even better than previous attempts at hypothesizing solutions. It seem applying the results to situations not concocted by oneself is the ideal method to achieve development, rather than responding to envisioned scenarios limited by ones' imagination. However, as the plan was to demonstrate the capabilities of the language through alterations of the case statement presented by Çeliku, the analysis chapter might have limited its demonstrative abilities, as interesting alterations to the same case quickly became exhausted. It is believed that by further exposing the concept not only to scenarios, but to the thought process of others, the development process would gain insight capable of taking it to a new level of usability.

Chapter 7

Future Work

As was the goal of the project, the design is not a complete language, and does not provide us with a functioning model. Such an ending leaves some quite obvious needs for future work with the canary manager, but also some undiscussed less protruding elements that might be work suitable to projects of a similar type as this.

A Control Entity

Having presented a variety of operators and functionalities, they still remain only as abstractions of the functions they intend to serve. At some point, these functionalities and features need to properly connect to an interpreter, or rather a compiler which enables them to serve their purpose. Like Begnum discussed his creation of the Managing Large Network (MLN) language, I envision something quite similar, where the language consists of key/value pairs, with the key serving as a trigger for a communicative entity, such as a daemon to distribute the paired value to its intended destination (Begnum, 2006). Admitting a lack of knowledge of what goes on ”under the hood” between a language interface and the resulting machine instructions, a full understanding of what this language necessitates is something that is missing. Nonetheless, with experience from a multitude of programming languages, I envision a member of the correct discipline will be able to connect the abstract dots to something more tangible.

Driver Entities

As boldly suggested, the manager would have a reliance on community based contributions to compose a library of drivers for tools and entities which would make increase the usability of the system, and increase the efficiency of implementation. To enable these contributions however, a standardized template for the driver must be accessible for the community to employ. Relying on the community for the success of the manager is arguably somewhat brash, and venturesome, and to combat the risks involved with pinning one's hopes on an open source project, one should aim to lower the threshold for making a contribution. This can be achieved by carefully designing the driver template. Researching methods, and finding a optimal solution which is both fault-proof, as well as understandable and easily created, will give the project the best chances of gaining traction.

Increasing Fidelity

Should a compiler or control entity as previously discussed see the light of day, the canary manager will be enabled to take further steps towards becoming a reality. Without a doubt, there is still plenty of operators, features and functions which will surface as needs for a system such as this, and it seems likely that by creating a prototype capable of employing the language, these needs will be uncovered. As was experienced by my own design process, attempting to apply the model to a use-case would almost instantaneously reveal either needs, or new potential for features, and I propose that with a system of higher fidelity, these revelations would continue to unfold.

Chapter 8

Conclusion

The goal of this project has been to achieve a model for a language capable of illustrating how one can efficiently handle the automation of canary deployments through high-level abstractions of complex situations. Central concepts for the language has been identified and discussed, and through them a series of features have been presented, both in explicit terms, as well as postulated possibilities.

The resulting design of this thesis shows capabilities in formulating shortened solutions to problems that would otherwise require more extensive scripting and preparations with making data available.

Though the language proposed does not stand fully completed, it demonstrates intended features and capabilities by applying it to model situations, and thereby answers to the problem statements and goals set in the preliminary chapters.

References

About usenix. (2017, 04). Retrieved 2022-04-20, from https://www.usenix.org/about

All conferences | usenix. (2022). Retrieved 2022-05-11, from https://www.usenix.org/conferences/all

Ambler, A., Burnett, M., & Zimmerman, B. (1992, 09). Operational versus definitional: a perspective on programming paradigms. *Computer*, *25*, 28-43. Retrieved 2022-04-12, from https://ieeexplore.ieee.org/abstract/document/156380 doi: 10.1109/2.156380

Begnum, K. (2006, 12). *Managing large networks of virtual machines* (Vol. 20). USENIX Association. Retrieved 2022-04-25, from https://www.usenix.org/legacy/event/lisa06/tech/full_papers/begnum/begnum_html/

Carzaniga, A., Fuggetta, A., Hall, R. S., Heimbigner, D., v., Wolf, A. L., & COLORADO. (1998, 04). *A characterization framework for software deployment technologies.* Retrieved 2022-04-25, from https://apps.dtic.mil/sti/citations/ADA452086

de Andrade, H. S., Schroeder, J., & Crnkovic, I. (2021, 08). Software deployment on heterogeneous platforms: A systematic mapping study. *IEEE Transactions on Software Engineering*, *47*, 1683-1707. Retrieved 2022-04-25, from https://ieeexplore.ieee.org/abstract/document/8786134 doi: 10.1109/tse.2019.2932665

Dearle, A. (2007, 05). Software deployment, past, present and future. *Future of Software Engineering (FOSE '07)*. Retrieved 2022-05, from https://ieeexplore.ieee.org/abstract/document/4221626 doi: 10.1109/fose.2007.20

Docker hub. (2022). Retrieved 2022-05-15, from https://hub.docker.com/

Elman, C., Gerring, J., & Mahoney, J. (2020). *The production of knowledge: Enhancing* *progress in social science*. Cambridge University Press.

Endres, C., Breitenbücher, U., Falkenthal, M., Kopp, O., Leymann, F., Wettinger, J., & Kopp, O. (2017). *Declarative vs. imperative: Two modeling patterns for the automated deployment of applications* (Doctoral dissertation). Retrieved from https://www.iaas.uni-stuttgart.de/publications/INPROC-2017-12-Declarative-vs-Imperative-Modeling-Patterns.pdf

Ernst, D., Becker, A., & Tai, S. (2019). *Rapid canary assessment through proxying and two-stage load balancing.* IEEE. doi: 10.1109/ICSA-C.2019.00028

Goldberg, K. (2012, 01). What is automation? *IEEE Transactions on Automation Science and Engineering*, *9*, 1-2. Retrieved 2022-05-01, from https://ieeexplore.ieee.org/abstract/document/6104197 doi: 10.1109/tase.2011.2178910

Implementation techniques for canary releases | gocd blog. (2021). Retrieved 2022-04-25, from https://www.gocd.org/2017/08/15/canary-releases/

Kaltenbacher, B. (2017). Intuitive interaction: Steps towards an integral understanding of the user experience in interaction design. *Goldsmiths Research Online*. Retrieved 2022-04-01, from https://research.gold.ac.uk/id/eprint/28677/ doi: https://research.gold.ac.uk/28677/1/CUL_thesis_KaltenbacherB_supp1_2008.swf

Lisa21 call for participation. (2021, 01). Retrieved 2022-04-20, from https://www.usenix.org/conference/lisa21/call-for-participation

McKenna, A. (2019). *Enabling invisible infrastructure upgrades with automated canary analysis.* Retrieved 2022-05-11, from https://www.usenix.org/conference/lisa19/presentation/mckenna

MV, T. (2019, 11). *What exactly is a programming paradigm?* freeCodeCamp.org. Retrieved 2022-05-01, from https://www.freecodecamp.org/news/what-exactly-is-a-programming-paradigm/

Nan, N., & Harter, D. (2009, 09). Impact of budget and schedule pressure on software development cycle time and effort. *IEEE Transactions on Software Engineering*, *35*, 624-637. Retrieved 2022-04-25, from https://ieeexplore.ieee.org/document/4815275 doi: 10.1109/tse.2009.18

Reiss, S. P. (1996, 03). Software tools and environments. *ACM Computing Surveys*, *28*, 281-284. doi: 10.1145/234313.234423

Rudrabhatla, C. K. (2020). *Comparison of zero downtime based deployment techniques in public cloud infrastructure.* doi: 10.1109/I-SMAC49090.2020.9243605

Schermann, G., Schöni, D., Leitner, P., & Gall, H. C. (2016). *Bifrost: Supporting continuous deployment with automated enactment of multi-phase live testing strategies.* Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2988336.2988348 doi: 10.1145/2988336.2988348

Searchtrends, G. (2015). *Google searchtrends.* Retrieved 2022-04-20, from https://trends.google.com/trends/explore?date=2017-01-04%202022-05-07&q=canary%20deployment,canary%20release

Singham, M. (1998, 09). The canary in the mine: The achievement gap between black and white students. *Phi Delta Kappan*, *80*, 8-15.

Sumina, V. (2021, 07). *26 cloud computing statistics, facts trends for 2022.* Retrieved 2022-03, from https://www.cloudwards.net/cloud-computing-statistics/

Uflacker, M., & Busse, D. (2022). Complexity in enterprise applications vs. simplicity in user experience. *Human-Computer Interaction. HCI Applications and Services*, 778-787. Retrieved 2022-04-20, from https://link.springer.com/chapter/10.1007/978-3-540-73111-5_87 doi: 10.1007/978-3-540-73111-5_87

Wurster, M., Breitenbücher, U., Falkenthal, M., Krieger, C., Leymann, F., Saatkamp, K., & Soldani, J. (2019, 08). The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Software-Intensive Cyber-Physical Systems*, *35*, 63-75. Retrieved 2022-04-20, from https://link.springer.com/

article/10.1007/s00450-019-00412-x doi: 10.1007/s00450-019-00412-x

Çeliku, L. (2021). *Towards continuity-as-code from local solutions to a high-level approach for automated canary deployments* (Doctoral dissertation). Retrieved from https://www.duo.uio.no/bitstream/handle/10852/87276/5/Lea_Celiku_NSA_Master_Thesis.pdf