Software Maintenance: An Alternative Production Environment and Other Improvements

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Timothy Cha

Fall, 2022

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Rosanne Vrugtman and Briana Morrison, Department of Computer Science

Software Maintenance: An Alternative Production Environment and Other Improvements

CS4991 Capstone Report, 2022

Timothy Cha
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
thc8pku@virginia.edu

Abstract

Cvent, an event-planning software-as-a-service company, needed maintenance on its Business Intelligence application as the application's microservices were undergoing several changes. One of the largest issues was the production environment in which the application's microservices were being tested, as the microservices could not be thoroughly tested without being deployed into the production environment and affecting the user directly. As an intern, I solved this issue by creating an alternative production environment, Prod-BETA, which allowed the microservices access to production-quality data without breaking the production environment. Steps that I took included sunsetting certain existing production environments and downsizing the memory requirements for each microservice in our application. Work was mostly performed on the IntelliJ IDE, with DataDog being used to analyze historical memory usage of each microservice. With the downsizing in memory and the removal of some production environments, the company saved a total of approximately $2000 worth of memory per month. Work is still in progress for completing Prod-BETA as at the time of writing, microservices are still being deployed to this environment by our full-time developers.

1. Introduction and Background

It is imperative that a company's application is continually changed to address evolving client requirements, security vulnerabilities, and potential bugs which can cause significant financial ramifications for the company. Code changes in earlier stages of software development, especially where those changes prevent possible bugs, substantially reduce the risk of unsatisfactory user experience and are less expensive to fix.

Cvent is no exception. Currently, Cvent's Business Intelligence (BI) Application has four deployment stages: local development, silo, staging and production. However, the Business Intelligence application's deployment cycle has an issue in that the silo and staging environments do not offer the high-quality data the production environment provides. This poses an issue because simulating the software's data in the silo and staging environments also does not guarantee that we will be able to catch all issues if we simulated the data in production. Therefore, the only way to test our production environment is by deploying

our code to the production environment directly.

However, this can negatively affect the user. We proposed restructuring the current problematic deployment environment architecture to introduce a new pre-production stage between staging and production called Beta. In that stage, we create an alternative environment named Prod-BETA which offers production-quality data without requiring testers to deploy the application's microservice code live.

2. Related Works

Former developer and cloud engineer Tomas Fernandez (2020) identifies two primary approaches to circumventing problematic releases into production: blue-green deployment and canary deployment. In blue-green deployment, developers perform side-by-side deployments with two identical production environments, referred to as Blue and Green. Both the Blue and Green environments have access to a shared resource, which includes databases and services. One environment (in this case Green) would be live for users to see while the other environment would not; instead, any versional changes to the source code would be tested on Blue. Once any errors caught during testing and deployment on Blue are mitigated, Blue will then become the live environment while Green becomes the staging environment. This cycle repeats. Canary deployment, on the other hand, rolls out software updates to only a select few users. Upon receiving feedback from those users, developers release the updated software to all users.

Several companies, such as Twitter for example, use canary releases to roll out updates in their software (Kotian, 2016).

Our restructured deployment architecture follows the blue-green deployment model more closely; however, it does not run two production environments simultaneously. Instead, Beta precedes our production stage sequentially.

3. Project Design and Methodology

We will improve the application's deployment architecture by adding an intermediate pre-production stage between staging and production.

3.1. Deployment Architecture Design

Below are figures for the deployment architecture before and after the change. The names in the figures have been changed to protect proprietary information.

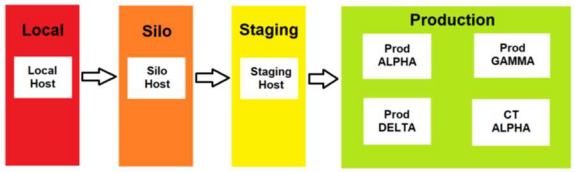

Figure 1: Cvent's BI Application's deployment architecture pre-change.

Prod-ALPHA, Prod-GAMMA, and Prod-DELTA are production environments, where Prod-ALPHA is currently the primary production environment for the BI services. These production environments differ by the region they operate in. CT-ALPHA is a continuous integration environment.

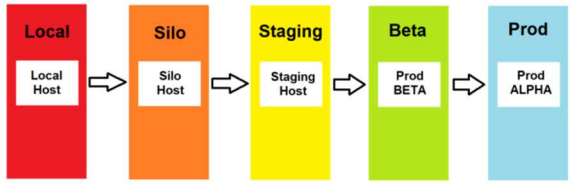

Figure 2: Cvent's BI Application's deployment architecture post-change.

Creating the new production environment was not an easy task, as many configuration files had to be edited or deleted, while several new ones were created.

3.2. Methodology

As part of restructuring BI's deployment architecture, many of the BI microservices needed readjustments on their memory requirements to lower the cost of running the application with this new architecture. To achieve this, I first used DataDog and AWS to analyze historical memory usage of each microservice. If a microservice's memory usage was significantly lower than its heap memory limit, I changed its limit to be closer to its memory usage while allowing some room for tolerance.

Moreover, the production environments Prod-GAMMA, Prod-DELTA, and CI-ALPHA were no longer being used, which necessitated their sunsetting for our new deployment architecture. To sunset those environments, their respective configuration files in each BI microservice were edited so that the number of microservice instances running in that environment would be zero. For example, if a prod-GAMMA.config file existed in the BI-service-alpha source code folder, the file would be edited so that zero instances of BI-service-alpha would run under the prod-GAMMA environment. Creation of the YAML and config files for Prod-BETA were handled by the full-time developers, and the migration status of all microservices from Prod-ALPHA to Prod-BETA is unknown at the time of writing because of the continuing work after the end of my internship.

Although not necessarily a component of our Prod-BETA project, the BI microservices required Software Development Kit updates to account for AWS outages as well as other security vulnerabilities as part of a remodeled deployment architecture. To do this, I edited each microservice's pom.xml file to update its mono-java version so that it used the latest Couchbase security patches. I also updated each microservice's Maven and dependency versions to their latest ones, and any dependency conflicts were mitigated by using exclusion tags.

4. Results

As part of our work, we as interns were asked by our manager to perform a cost analysis on the BI application pre- and post-change in our deployment architecture. With the memory readjustment for every BI microservice across the silo, staging, and production environment Prod-ALPHA, approximately 150 GB was saved. Moreover, the cost of running the application with the right-sizing of the BI microservices and the sunsetting of Prod-GAMMA and Prod-DELTA was reduced by about 65% per month. This substantial decrease in cost is financially significant to the company as we do not have to spend as much money running instances of our microservices on AWS.

5. Conclusion

Cvent's Business Intelligence application had issues with its production environment where in that testing the application's microservices in that environment would directly affect the user. Because testing the microservices in the production environment provided higher quality data than testing in lower environments would provide, changes needed to be made to the application's deployment architecture. This was accomplished by creating Prod-BETA, an alternative pre-production environment,

which allowed access to production-quality data. Creating Prod-BETA involved many steps from sun-setting certain production environments to down-sizing the memory requirements of the application's microservices. In the end, creating Prod-BETA saved the company several thousand dollars per month on memory usage.

6. Future Work

At the time I completed the internship, full-time developers were still in the process of creating the Prod-BETA YAML files necessary to deploy the software in the Prod-BETA environment. I do not know whether this step was completed. In the future, the deployment architecture of the Business Intelligence application may be changed again while still maintaining the ability to obtain high-quality data to ensure an optimal user experience.

References

Fernandez, T. (2020, August 5). What Is Blue-Green Deployment? Semaphore. https://semaphoreci.com/blog/blue-green-deployment

Fernandez, T. (2020, September 1). What Is Canary Deployment? Semaphore. https://semaphoreci.com/blog/what-is-canary-deployment

Kotian, A. (2016, May 9). Manhattan software deployments: How we deploy Twitter's large scale. Retrieved from https://blog.twitter.com/engineering/en_us/topics/insights/2016/manhattan-software-deployments-how-we-deploy-twitter-s-large-scale-distributed-database