Heartland Escapes: Architecture Strategy

CS644 - Computer Systems Architecture

Aidan Polivka

August 4, 2024

Revisions

Number	Date	Description
1	07/28/2024	Initial Creation
2	08/04/2024	Resolve instructor feedback: Reference hanging indents
3	08/04/2024	Development Processes installation

Table of Contents

1. Project Outline

Heartland Escapes is a bookstore that's been steadily gaining popularity in Nebraska. Their store has roots in Lincoln Nebraska and has expanded to three additional locations (two total locations in Lincoln, one in Omaha, one in Grand Island). A major reason for Heartland Escapes' popularity as a bookstore is the events that they host in store. Advertising for these events on social media has resulted in a semi-viral response, which has funded additional technological development for the company. Heartland Escapes is a relatively small company, but they have a competitive culture and a forward-thinking leadership crew that continues to look for technological solutions to compete with bookstore giants like Barnes & Noble.

Heartland Escapes has a small IT department now that they've developed so many in-house software solutions. They've decided that it would be best to retain three software engineers and the CIO and use contractors to support larger scale development efforts. This allows for constant technical support and maintenance for the company, and native first-hand knowledge of how the Heartland Escapes systems work without needing to retain a large development force.

Heartland Escapes has undergone a substantial transformation over the past few years. They migrated their point-of-sale system from on premises to Google Cloud, then conducted a large development effort to integrate company administrative processes into the point-of-sale system. So now that system is a hub for all internal processes to Heartland Escapes, including employee time clock, inventory reporting and re-ordering, accounting reporting, and inventory and product management.

Now that those projects are complete, Heartland Escapes is ready to move forward with development of their very own e-commerce platform. This platform will require integration into their pre-existing inventory and accounting systems. This architecture strategy is intended to support this development effort, resulting in a fully qualified architectural design for both the software and infrastructure of this e-commerce website.

2. Software Architecture and Evaluation

2.1 Existing Software Architecture

The pre-existing software solution at Heartland Escapes is their point-of-sale system. This system has been adapted to support additional administrative activities within Heartland Escapes. The primary components to this system include a .NET MVC application that services the client application and two supporting REST APIs: the Inventory API and the Accounting API. These APIs are both written in .NET Core C# and have their own SQL Server databases, an accounting_db and an inventory_db. The .NET MVC application also has a database that hosts data like employee timecards and store event schedules.

The infrastructure for the point-of-sale system is relatively simple. The APIs and MVC app are all deployed in GCP's Cloud Run environment. Cloud Run is a fully managed google cloud service, meaning that the infrastructure for these applications does not need to be maintained. Similarly, the SQL Server databases are hosted in Google's Cloud SQL components which are almost fully managed.

2.2 Architecture Strategy Objectives

It's important for Heartland Escapes to be able to integrate this new e-commerce solution into their pre-existing APIs for tracking and management of inventory and sales. Additionally, the maintenance burden on their development team would be greatly lessened by continued use of familiar technologies. Finally, since Heartland Escapes is a small company, we'll want to find the most cost-effective solution that requires the least amount of infrastructural maintenance.

It's also important to bring in the perspective of the domain when choosing an architectural solution. The domain of e-commerce is typically event based. Consider what needs to occur after a user checks out their cart:

1. The customers card must be processed
2. If their payment is successful, available inventory for the items purchased must be decreased
3. The customer must receive an email confirmation of their order with an invoice
4. If the items purchased reach the restock threshold, a new order must be sent to that product's supplier
5. The customers order must go to order fulfillment, where it's packaged and shipped to their home

E-commerce is also usually easily componentized. Consider all the different sub-processes within e-commerce: Inventory management, payment processing, order fulfillment, shipment tracking,

shopping cart management, product display, filtering, and recommendations, etc. Additionally, e-commerce applications typically require complex user interfaces to support these pieces of functionality.

2.3 Infrastructural Architecture Options

There are three primary options for infrastructure based on the preliminary requirements outlined in section 2.2. For the sake of simplicity and cost savings, all solutions will be deployed to Cloud Run. Since Cloud Run is fully managed, pay per use and can scale to zero instances, this component is the best option for Heartland Escapes regardless of architecture solution.

2.3.1 Microservices Architecture

A microservices architecture would allow for the development of loosely coupled units of domain functionality. Each process can be broken into its own modular set of code, providing a suite of benefits. This will allow engineers to work asynchronously on different microservices without bumping elbows and enforce the separation of responsibilities between software components. This solution also provides a highly robust and independently scalable solution. If one microservice node fails, it doesn't necessarily mean the whole system is down. Also, if the product catalogue is receiving high traffic volume, product catalogue service nodes can scale independently of the payment processing service nodes. Finally, the microservices architecture can easily send requests to the inventory and accounting APIs, fulfilling that requirement for Heartland Escapes.

This would also include a separate single page application to support the user experience. The framework used will be up for debate later in this architecture strategy.

Although the microservices architecture offers many benefits, there are also complications. Because microservice communication is network based, latency may be an issue for processes that need to run through multiple services. Also, microservice architecture is a complex solution to implement that may put a strain on the maintenance teams. Without careful configuration and documentation, this solution could result in failure. (IBM, n.d.)

2.3.2 Modularized Monolith Architecture

The term "monolith" tends to scare individuals because it is not as progressive a solution as microservices. When many engineers think of monoliths, they think of tangled up legacy applications that are not well modularized. However, with strong development practices and guidelines, a monolith is still a completely viable architecture with today's technology (Belcher, 2020). This solution would extend the existing point-of-sale application to support the e-commerce system. Each sub-domain listed in section 2.2 would have its own module (likely a C# project) in code. This separation of concerns will ensure that everything is written in the same language and is simple to maintain for the IT folks at

Heartland Escapes. Because this solution is enclosed within a single application, inter-process communication latency will be significantly reduced.

Also, a monolithic solution could drastically impact the feature development speed of the team in a positive way. By developing within a monolith, full vertical sliced features can be developed at a time without needing to worry about developing additional infrastructure like REST calls, CI/CD pipelines, or Pub/Sub topics and subscriptions. (Ozkaya, 2023)

This solution has its pitfalls as well. Although the architecture is simple and easy to maintain, it is not as robust as the microservices architecture. If a part of the system fails, it may break the entire system. Additionally, sub-processes within the system are not independently scalable, potentially leading to increased operational costs.

2.3.3 Hybridized Solution

Because the point-of-sale MVC system is already a monolithic core, that core could be utilized for common functionality in the e-commerce system like user management and product display. This core would support the core functionality of Heartland Escapes e-commerce system including serving the client application, while microservices handle the other application functions like notifications, payment processing, order fulfillment, etc. This would provide a happy medium between the modularized monolith, and the microservices architecture.

2.4 What's the Best Architectural Option?

The best solution for Heartland Escapes is going to be 2.3.3, the hybridized solution. This provides the positives between the modularized monolith and microservices and softens the negatives. This should result in a more easily maintained system for Heartland Escapes, along with a quicker turn around on feature development and a shorter timeline on product launch.

3. Development Processes

3.1 Current Processes

The permanent members of Heartland Escapes development team use a Kanban development system to manage their work backlog and support their maintenance efforts. Kanban works well for small scale development efforts, and particularly well with maintenance work. Kanban focuses on visual workflow management to ensure that work is not lost, work in progress limits to ensure that features are delivered, and continuous delivery to maintain a steady flow of feature delivery and issue resolution.

This project management method will work well for a small team focused on maintenance. With the small team, they can allow members of the company to submit feature requests and bug fix requests to a Kanban backlog. Those requests can be filtered by the CIO, assigned priority, and delivered to the development team. Development team members can pull items out of the backlog as they see fit but are limited to the number of items they can pull into “work in progress” at a time. Then the CIO can run metrics on the board for burn down and estimates sprint by sprint.

The development team participates in Scrum ceremonies like sprint planning, sprint reviews and daily standups. Other ceremonies are held on an as-needed basis, like backlog refinement and retrospectives. The CIO acts as scrum master for the development team.

The development team currently uses CI/CD pipelines to deploy their applications automatically on merge to the devdeploy branch. Once the feature is dev tested and approved by another team member, it’s merged into the main branch and automatically deployed to the Sys environment for user acceptance testing by the CIO. Prod releases are manually deployed on a biweekly basis from the main branch. (Martins, 2024)

3.2 Suggested Processes

A simple Kanban process will not fulfill all the required processes for a large-scale development effort. However, there are Agile development methods that extend Kanban and Kanban principles to better support large development efforts. One such development method is called Scaled Agile Framework, or SAFe. SAFe takes similar principles from traditional Kanban, including the visual workflow management, work in progress limits, and continuous delivery, but scales it up to fit a large development effort.

An oversized product delivery team tends to suffer from siloed knowledge. In a traditional development environment with a large crew, it can be easy for a single individual to be the only person how knows how a part of the system works. SAFe tries to resolve this issue by creating diverse teams

within the organization. For example, rather than there being a DevOps team, a Data team, and an Application Development team, there would be multiple teams each with individuals who can support DevOps, Data, and Application Development. This structure is called an Agile Release Train, or ART. With this development structure, the permanent employees at Heartland Escapes can remain exposed to the entire software stack during the development process. There would be multiple Agile Release Trains working in parallel on different epics (Agile Release Train, 2022). At this scale of a development effort, all scrum ceremonies are recommended to further refine the process. Kanban and SAFe are both proponents of continuous improvement, which is fueled by scrum ceremonies like Sprint Review and Sprint Retrospectives.

SAFe uses a Kanban board for backlog and work visualization. However, that board is populated by processes not typical to traditional Kanban. There is a role in SAFe called an Epic Owner. This individual is usually a product owner, system or enterprise architect, project manager or program director. Their job as an Epic Owner is to collaborate with the stakeholder to farm epics. In these sessions with the stakeholders, the Epic Owner is responsible for defining the epic, the lean business case, and the definition of the minimum viable product. They are then responsible for carrying that epic through the development process. (Epic Owner, 2023)

The suggested deployment process is going to be the same as the current deployment process. One of the two changes that should be made is who the approval gate is for Sys and Prod. The solution architect or tech lead should approve changes to deploy in Sys, and the user acceptance testers should approve changes to deploy in Prod. The other change would be to include automated code quality gates to deployment pipelines. This includes strong integration testing coverage between services, ensuring that changes made between teams don't break service to service contracts. Additionally, unit test code coverage gates, software linting rules, vulnerability scanners and image attestation & authorization actions will be required.

3.3 Architecture

The development processes must suit the architecture Heartland Escapes is developing. To recap the suggested architecture, it is recommended that Heartland Escapes takes a hybrid approach by extending the monolithic .NET MVC application for shared functionality between the point-of-sale system and the e-commerce site and to utilize microservices for the new stateless domain functionality of the e-commerce site. An important note is that the monolith .NET MVC application should be extended with a focus on modularity and reuse, creating a "modular monolith".

With the focus on modularity and extensibility, an iterative Agile approach is key to supporting this architecture. The iterative nature of Agile's requirements gathering process requires that the system maintains its ability to be adaptive. If Heartland Escapes chooses to assign an ART per microservice, the changes made to the CI/CD pipelines for automated integration testing will be incredibly important to

supporting this architecture and team structure. This method of assigning an ART per microservice also decentralizes the decision-making process, allowing teams to be self-organizing and self-managed. This will support asynchronous development of the e-commerce site, resulting in increased velocity.

The incorporation of an Epic Owner can help facilitate Domain Driven Design within the system. Having an individual take ownership of an Epic from inception with the stakeholder to delivery with the development team will help foster domain expertise within the larger team. This will also support the collection and use of ubiquitous language between stakeholders and the project delivery team. Also, the ARTs should be working within bounded contexts, supported by the microservice architecture. (Stemmler, 2019)

An architectural addition that may support this environment could be clean architecture. The premise of clean architecture is to separate presentation, domain, and infrastructure logic into independent layers. All database connection, third party integration, and API to API communication should be handled in the infrastructure. Domain functionality and business logic should be handled in the domain layer. API configuration and client-side code should be handled in the presentation layer. These layers should be separated within the file structure, and only available between each other using dependency injection. Having the domain logic centralized and well-defined strongly supports Domain Driven Design. Clean architecture also ensures that each layer (presentation, domain, and infrastructure) is tested independently which facilitates continuous integration and deployment from SAFe. The loose coupling of these layers supports better interoperability, a tenant of Agile development. (Jacobs, 2020)

4. Architectural Design Strategy

TBD

5. Engineering Requirements

TBD

6. Emerging Technologies

TBD

References

Belcher, M. (2020, September 8). *Breaking Down the Monolith*. Codurance. https://www.codurance.com/publications/2020/09/08/breaking-down-the-monolith

IBM. (n.d.). *What are Microservices? | IBM*. Www.ibm.com. https://www.ibm.com/topics/microservices

Ozkaya, M. (2023, April 16). *Microservices Killer: Modular Monolithic Architecture*. Medium. https://medium.com/design-microservices-architecture-with-patterns/microservices-killer-modular-monolithic-architecture-ac83814f6862

Agile Release Train. (2022, October 24). Scaled Agile Framework. https://scaledagileframework.com/agile-release-train/

Epic Owner. (2023, June 30). Scaled Agile Framework. https://scaledagileframework.com/epic-owner/

Jacobs, J. (2020, February 19). A Brief Intro to Clean Architecture, Clean DDD, and CQRS. Jacobsdata.com. https://blog.jacobsdata.com/2020/02/19/a-brief-intro-to-clean-architecture-clean-ddd-and-cqrs

Martins, J. (2024, January 19). What Is Kanban? A Beginner's Guide for Agile Teams [2023] • Asana. Asana. https://asana.com/resources/what-is-kanban

Stemmler, K. (2019, November 5). Comparison of Domain-Driven Design and Clean Architecture Concepts | Khalil Stemmler. Khalilstemmler.com; khalilstemmler.com. https://khalilstemmler.com/articles/software-design-architecture/domain-driven-design-vs-clean-architecture/