GETTING STARTED WITH HEADLESS COMMERCE

ON SALESFORCE B2C COMMERCE

Introduction

Going headless with Salesforce B2C Commerce gives you the flexibility to interact with your customers on multiple touchpoints using simple yet powerful APIs that perform and scale to meet your growing business needs. The Salesforce B2C Commerce APIs make it easy for your engineering teams to focus on the front-end user experience while leaving the heavy lifting to us when it comes to programmatically interacting with your digital ecommerce resources. If you're completely new to headless or still deciding whether headless is right for you, consider reading this blog post: Everything You Need to Know About Headless Commerce (But Were Afraid to Ask).

The Salesforce B2C Commerce APIs have been productized to provide even easier and more powerful integrations with your custom applications that perform and scale even beyond the level you've come to expect from Salesforce B2C Commerce. For example, with Salesforce Einstein artificial intelligence (AI) built right into our APIs, you won't need to hire a team of data scientists to ensure your users are always getting the most relevant recommendations to help boost your revenue.

Whether your storefront already uses the full Salesforce B2C Commerce stack or not, headless is becoming an important consideration as your customers engage in more commerce outside of the traditional online storefront. Some of our customers choose to build their digital online storefront using 100% headless architecture while others use a traditional Salesforce B2C Commerce implementation based off of our Storefront Reference Architecture (SFRA) – with headless implementations for other touchpoints such as mobile/social applications, smartwatch integrations, or even connected vehicle integrations. Regardless of which touchpoints you've chosen to go headless, the high-level architecture involved in the integration with Salesforce B2C Commerce will not vary significantly.

> *With a custom head, you have the flexibility to be creative in order to provide the unique and tailored front-end user experience that your customers have come to associate with your brand.*

Taking on this challenge is not for everyone since building, hosting, and maintaining high-traffic public-facing applications can be extremely challenging. There are a number of considerations we hope to help your team understand when it comes to maintaining the same level of predictable performance and scale offered by the Salesforce B2C Commerce platform.

Contents

1. Roles 04

Solutions Architect 04

Developer 04

Quality Assurance (QA) Engineer 04

Security Engineer 04

Site Reliability Engineer (SRE) 04

Operations Engineer 04

2. Architecture Overview 05

3. Considerations for Deploying and Maintaining a Headless Architecture 06

Content Delivery Network (CDN) 06

Scaling 07

Caching 08

Resilience, Redundancy, High Availability, and Disaster Recovery (HADR) 09

Middleware and Operating System Configuration 10

Networking 10

Serviceability (Logging, Alerting, Monitoring, Troubleshooting) 11

Security 11

Bot Detection/Mitigation (PerimeterX, Cloudflare, Akamai Bot Manager) 12

Load Testing 12

Penetration Testing 12

4. Conclusion 13

5. Appendix 14

1 Roles

Building and maintaining a headless solution requires the following high-level roles in your or your partners' organization. Skillsets may overlap in these roles, and your organization may define them slightly differently.

Solutions Architect

The solutions architect designs the technical solution and oversees the development from both a technical and process perspective.

Developer

The developer is responsible for delivering a functional product that meets the requirements laid out by the solutions architect. Your custom head will likely consist of both front-end applications that provide the presentation layer as well as a back end that serves content to the front end. A front-end developer will focus on building the front-end user interface (UI), while a back-end developer might deliver an API client for the front-end developer to leverage from the UI. It is also possible for an experienced full stack developer to work on both the front end and the back end.

Quality Assurance (QA) Engineer

The quality assurance engineer is responsible for validating the quality of the application stack, including functional, performance, and resilience characteristics. The QA engineer will define the test suite and execute it against the application after each build to ensure no functional regressions have been introduced with new code. The QA engineer will also design requirements for load testing and advise the technical team on where certain code might require performance optimizations.

Security Engineer

The security engineer is responsible for protecting the network and applications from users with malicious intent. The security engineer will also oversee security-related tests – for example, penetration testing or source code vulnerability scans. The security engineer also assists in investigations related to potential security-related incidents.

Site Reliability Engineer (SRE)

The site reliability engineer (SRE) is responsible for the overall health and performance of the application stack. The SRE provides emergency response to production incidents and also builds systems to monitor and maintain the reliability and performance of your applications.

Operations Engineer

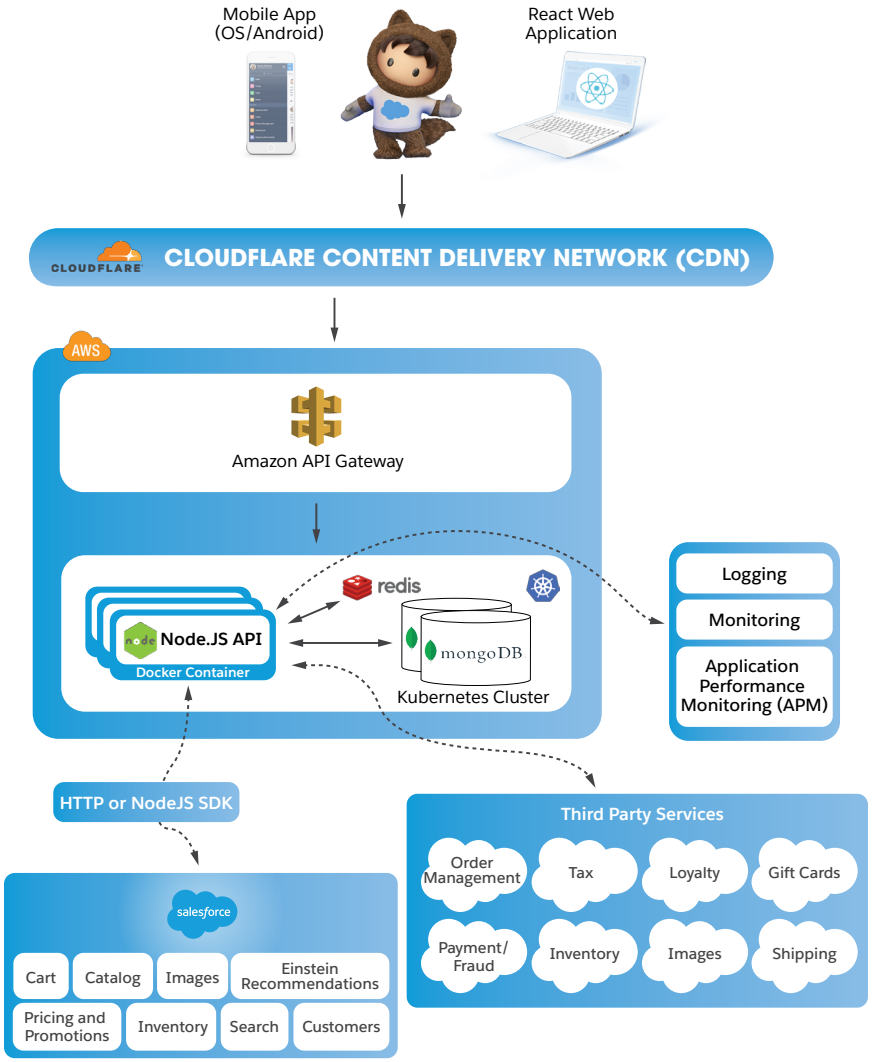
The operations engineer has expertise in systems and network engineering. These engineers will be involved in designing and maintaining operating systems and middleware configuration, patching software and systems, and maintaining/troubleshooting network-related issues.

2 Architecture Overview

Your custom head will serve as the front-end presentation layer, providing a rich digital shopping experience for your customers. The head interacts with Salesforce B2C Commerce APIs to provide the user with digital content and functionality to complete their purchase from a wide range of devices. The Salesforce B2C Commerce APIs provide the functionality to easily and securely interact with all of your digital content while your custom head can connect to third parties to allow the user to complete their transaction.

You will have the flexibility to develop your custom head using a combination of industry-standard open-source technologies, home-grown frameworks, or Salesforce-guided solutions that use technologies like Heroku for managing and deploying your runtime. In the end, your goal is to provide your users with the best front-end experience possible while maintaining a highly performant and scalable system no matter the load level.

Example of Salesforce B2C Headless Implementation



See the Appendix section for an overview of other technologies that might also be used in a headless implementation.

3 Considerations for Deploying and Maintaining a Headless Architecture

The Salesforce B2C Commerce platform provides a highly reliable, scalable, and performant solution to ecommerce. A number of components contribute to our success, including a dedicated 24/7 operations team, networking and security experts, performance experts, database experts, load-testing experts, an embedded content delivery network (eCDN), highly tuned middleware components, proactive monitoring/alerting, and much more.

> *Many of these components and areas of expertise do not come "out of the box" in a headless implementation and will be required within your or your partners' organization as you develop your head for the same level of performance and scale you have come to expect from the Salesforce B2C Commerce platform.*

Content Delivery Network (CDN)
The Salesforce B2C Commerce platform leverages an eCDN to serve cached static content such as product images, JavaScript, and HTML from a point of presence (PoP) server closest to the user (otherwise known as the "edge") in order to ensure the fastest content delivery possible.

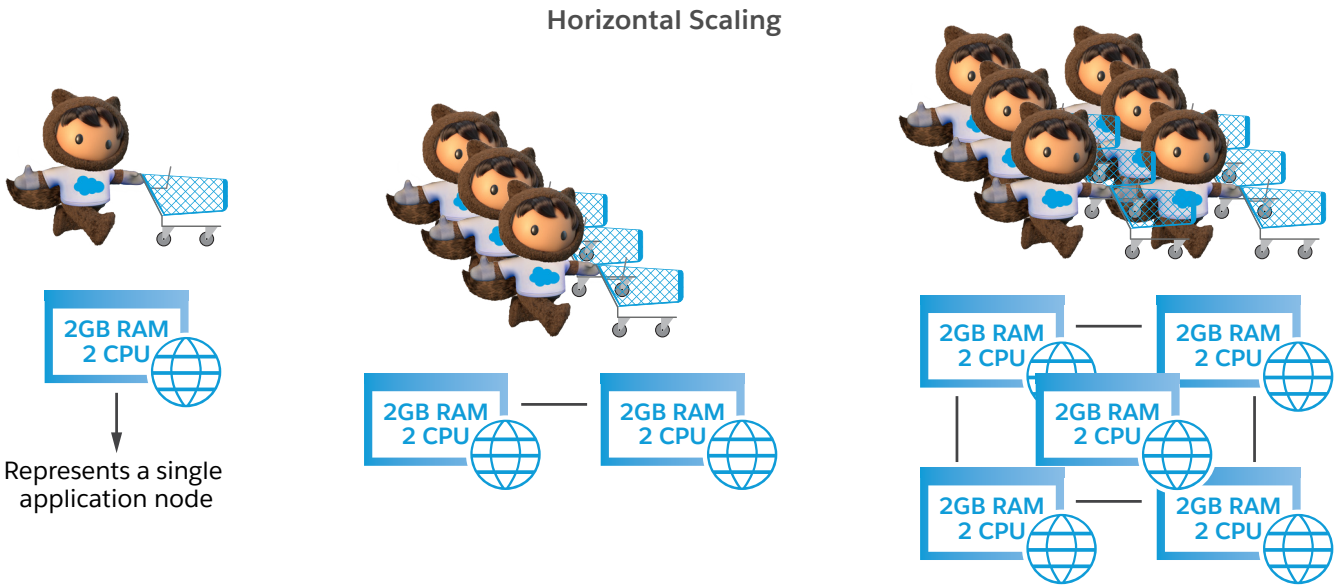
You will need to ensure you are adopting a content delivery network (CDN) to serve cached content from your custom head back to the user. Doing so will prevent unnecessary requests to your applications and infrastructure as content gets served from the edge, allowing you to only process requests that actually require real-time processing. This allows for better performance and scalability of your custom head. Many CDNs also provide other performance-optimization-related features such as code minification, bot detection/mitigation, firewall security, and advanced routing capabilities.

Scaling

The Salesforce B2C Commerce platform uses scaling techniques that ensure your storefront remains available and equally performant during periods of heavy traffic. As system resource utilization increases (often due to a significant increase in traffic), additional application nodes are often added to the cluster to ensure the performance of your storefront remains unchanged. This technique is referred to as horizontal scaling.

When it comes to the custom head, it is important to implement similar strategies to ensure your applications scale to meet the demands of customer traffic. Many cloud providers offer features for defining autoscaling policies that will provision additional applications when necessary in the event key metrics such as memory and CPU utilization have breached predefined thresholds. You will need to understand the boundaries of your application stack to determine the ideal autoscaling triggers to ensure your application can withstand sudden bursts of traffic.

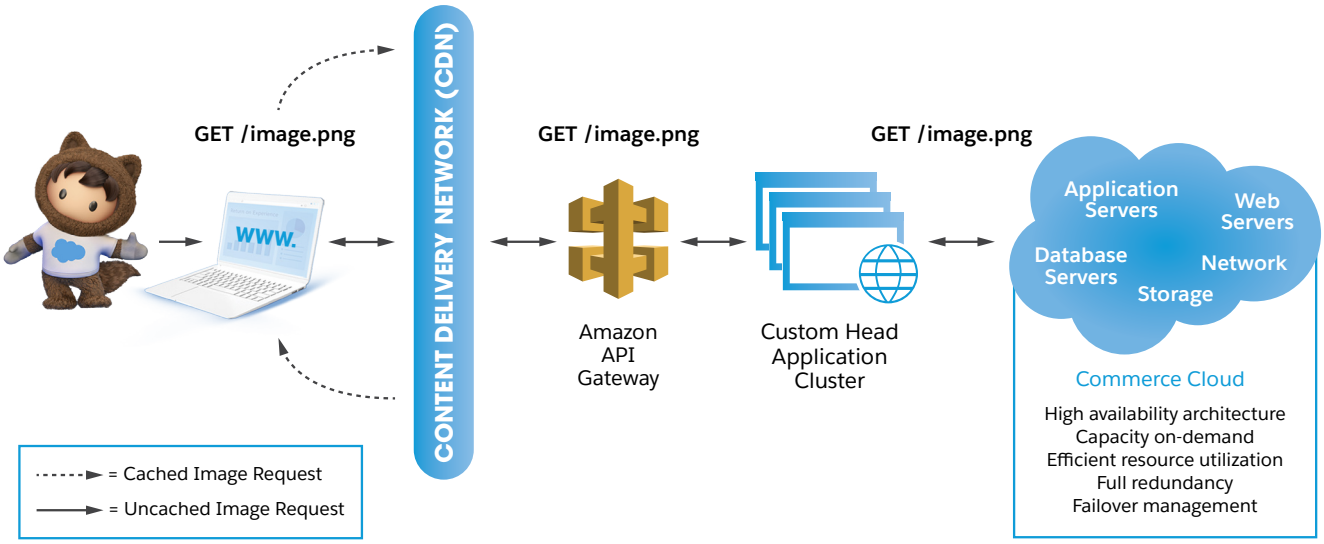
If your application stack is deployed using containers, you will need to understand container orchestration. This is handled by Kubernetes in our high-level architecture diagram in the Architecture Overview section. These tools will help with application provisioning, resource balancing among a cluster of servers, health monitoring, load balancing, and more.



Caching

Caching is one of the most important aspects of a highly performant and scalable system, as it reduces the need for a request to travel all the way from the client's browser to the origin server before returning content back to the user. With multiple layers involved in a headless ecommerce architecture, it is important to return data back to the user at the closest layer whenever possible. As mentioned in the preceding section, scaling an application cluster horizontally lets multiple applications process requests when you need more processing power, but this can still result in a poorly performing system if caching is not properly implemented throughout your applications and infrastructure.

The diagram below depicts an example of a cached image request being served by the CDN as well as an uncached request traveling all the way through the infrastructure to retrieve an image. For a single image, this could mean the difference between a 30ms response time and a 500ms+ response time. **Caching content at every layer will greatly improve the overall response time of your application while also taking the strain off of your back-end applications.**



You'll need to determine which types of requests should be cached throughout the various layers of your infrastructure to ensure the fastest possible page-load times, as well as the duration for which resources will be cached. Other types of caching to consider include:

1. **Distributed caches** such as Redis that are shared across application nodes for the purpose of reducing the load on your underlying storage systems. For example, you might choose to cache an expensive API query result in a distributed cache to prevent each application in a cluster from issuing the same request multiple times.

2. **In-memory caches** that live within your applications' physical memory that serve to avoid expensive calculations within your application. These types of caches are usually provided by your application runtime SDK or as supplemental libraries.

3. Salesforce B2C Platform caching within Salesforce B2C Commerce APIs to ensure that a request already processed by one user does not need to be recalculated for another – as long as the requested data is not unique across users. Our APIs make it easy for developers to cache requests with granularity to ensure flexibility and optimal performance.

Resilience, Redundancy, High Availability, and Disaster Recovery (HADR)
Salesforce didn't get their best-in-class reputation by having single points of failure. Not only does every component within the Salesforce B2C Commerce infrastructure have redundancy in the event of single node failures, we are also prepared for disasters with standby systems running in separate data centers to minimize interruption should disaster strike. We also design our applications to fail fast in the event of known service degradations. For example, in our traditional storefront implementation, we prevent calls to third parties if we know their service is performing poorly. Doing so prevents application threads from being tied up for long periods of time in order to avoid a performance degradation or unavailability. You'll retain this quality of service when calling Salesforce B2C Commerce APIs from your custom head, but you will need to think about how you'll provide the same level of service within your custom head.

> *Plan for both single node failures as well as entire infrastructure outages to ensure your application remains available in the event of a disaster.*

Most modern cloud providers build high availability and disaster recovery into their offerings. However, it is important to understand how a single node failure can impact your application. For example, if you are load-balancing between two web servers to serve traffic coming into your application, make sure you understand the impact to your application should one or both web servers become unavailable, especially during periods of heavy traffic. Furthermore, make sure you have a recovery plan for these situations, whether you are using autoscaling policies to provision new applications or having a 24/7 team in place to quickly react in the event of a partial or total service disruption.

You will need to design your applications to be resilient enough to handle service disruptions gracefully without causing negative user experiences. For example, if your loyalty service becomes extremely slow to respond to requests or becomes unavailable, consider implementing circuit breaker logic that blocks calls to the point balance check when a user logs in to your application until the service recovers. A user who cannot see their loyalty balance after logging in is more likely to continue shopping than a user who can't log in at all. Many cloud providers offer flow control/circuit breaker functionality within their API gateways, and there are libraries such as resilience4j for building the same functionality into your custom back-end systems.

Middleware and Operating System Configuration
The Salesforce B2C Commerce platform is comprised of many different middleware components that are configured for optimal performance and scale to suit our entire customer base. Tuning middleware and supporting components such as web servers, relational database management systems (RDBMS), runtime containers such as Docker, operating systems, and even custom application parameters can be a time-consuming process, often involving trial and error through load testing before reaching the optimal configuration.

> *Not only do you need to find the middleware architecture that best suits the needs of your custom head, you will need experts within your or your partners' organization who have a good understanding of middleware deployment, maintenance, and configuration.*

A highly performant and scalable application can still suffer a performance degradation if middleware and supporting systems are not correctly configured or maintained. For example, performance of an application that interacts with an RDBMS might degrade due to index fragmentation if routine database tasks are not performed regularly.

Networking
Our network team within Salesforce B2C Commerce is extremely knowledgeable about how traffic flows through our network and infrastructure and ultimately into our point of delivery (POD) hosting our Salesforce B2C Commerce applications. We are able to quickly troubleshoot network issues should they arise, even if the issue resides outside of our network.

> *Understanding the networking components related to your custom applications and infrastructure will be paramount to your success.*

Application components deployed on different networks across different cloud providers can be extremely difficult to troubleshoot when network issues arise. Consider this when designing your system architecture and ensure you have network experts in-house who are capable of capturing and reviewing network traces when the network comes into question due to performance or unavailability-related problems.

Serviceability (Logging, Alerting, Monitoring, Troubleshooting)
At Salesforce B2C Commerce, we build our applications not only for performance and scale, but also to provide valuable data to our support and operations teams about application behavior in the event of a problem. Not only do we have a flexible logging and monitoring/alerting framework in place, we also provide applications for visualizing system and application logs, which allow us to quickly highlight common problem areas of our applications before a user would ever notice something is wrong.

> *We highly recommend defining your logging and monitoring framework requirements prior to building your custom applications to avoid rewriting code to facilitate logging and monitoring.*

Consider using application performance monitoring (APM) tools such as AppDynamics that interrogate your application runtime environments to populate analytics data without introducing significant performance overhead. These tools will make it much easier to identify the source of a problem. Many APM tools also provide alerting capabilities that will let your support and operations teams know as soon as a potential problem has been detected. These problems could include an increase in error messages, sustained increases in server level metrics, or even specific application function calls performing worse than normal.

Built-in debugging capabilities exist in most application runtime SDKs that make it easy for you to periodically collect diagnostic data from your applications. This data can be extremely valuable if collected regularly and retained for some period of time. It can enable you to troubleshoot problems after they've occurred, rather than having to wait for the problem to reoccur while you debug in real time.

Security
There's nothing Salesforce takes more seriously than securing our applications and our customers' data. At Commerce Cloud, we regularly patch every component that makes up our platform to keep the risk of security-related incidents as low as possible.

> *The responsibility of security comes with running your own infrastructure and application stack.*

Many cloud providers take care of patching operating systems and middleware, but security vulnerabilities can also exist within your custom applications' third-party libraries. Ensure you have security experts in-house who have a plan for routine software patching, and ensure developers are trained to take appropriate security measures when building your custom applications. Your security experts should be able to monitor and review application logs in the event of a potential security-related incident and understand the wide variety of web-based vulnerabilities. We recommend running security scans and penetration tests on test/sandbox instances in order to eliminate the risk of corrupting production data in the event the penetration tests are successful in modifying your application data.

Bot Detection/Mitigation (PerimeterX, Cloudflare, Akamai Bot Manager)

Malicious bots are a real and growing threat to ecommerce websites. Since you will be running your head outside of Commerce Cloud, it is important to consider the various endpoints that might be abused by malicious bots and understand how you can protect yourself against them. Implementing CAPTCHAs on login pages is just one technique used to protect against brute force attacks, but there are many common types of attacks that are often not considered until it's too late. Read this whitepaper by PerimeterX to get a clear understanding of the common types of attacks launched against ecommerce sites and learn what you can do to protect yourself against them. Consider techniques offered by your CDN, such as Rate Limiting on commonly abused endpoints such as login pages and gift card/loyalty balance checks, and consider blocking requests to sensitive endpoints that do not contain headers that you would expect to see coming from a real user's browser.

Load Testing

Load testing is a critical component of the success of your online Salesforce B2C Commerce Storefront, whether you are running your own head or not. Obvious application performance issues may be resolved prior to load testing. However, there are likely scenarios you did not consider that may result in poor performance when your application is under heavy load. Consider using a partner to conduct and deliver results of your load test to ensure your application can scale to handle your projected traffic forecasts. Load test reports will highlight weak spots in your application that did not perform as expected under load. Most of our headless customers load test the Salesforce B2C Commerce integrations independent of site functionality provided by the custom head. Using this approach, you'll be able to differentiate between a performance problem related to Salesforce B2C Commerce and a performance problem involving your custom head.

Penetration Testing

Penetration testing is the practice of executing scripts against your application that test for known security vulnerabilities. This is an important activity to conduct throughout your development cycle as it will identify security vulnerabilities in your applications that could allow an attacker to gain access to sensitive back-end data through the front end. Penetration testing applications are destructive by their nature, so always make sure to only run penetration tests in environments that can easily be rebuilt, such as a sandbox or test environment.

4 Conclusion

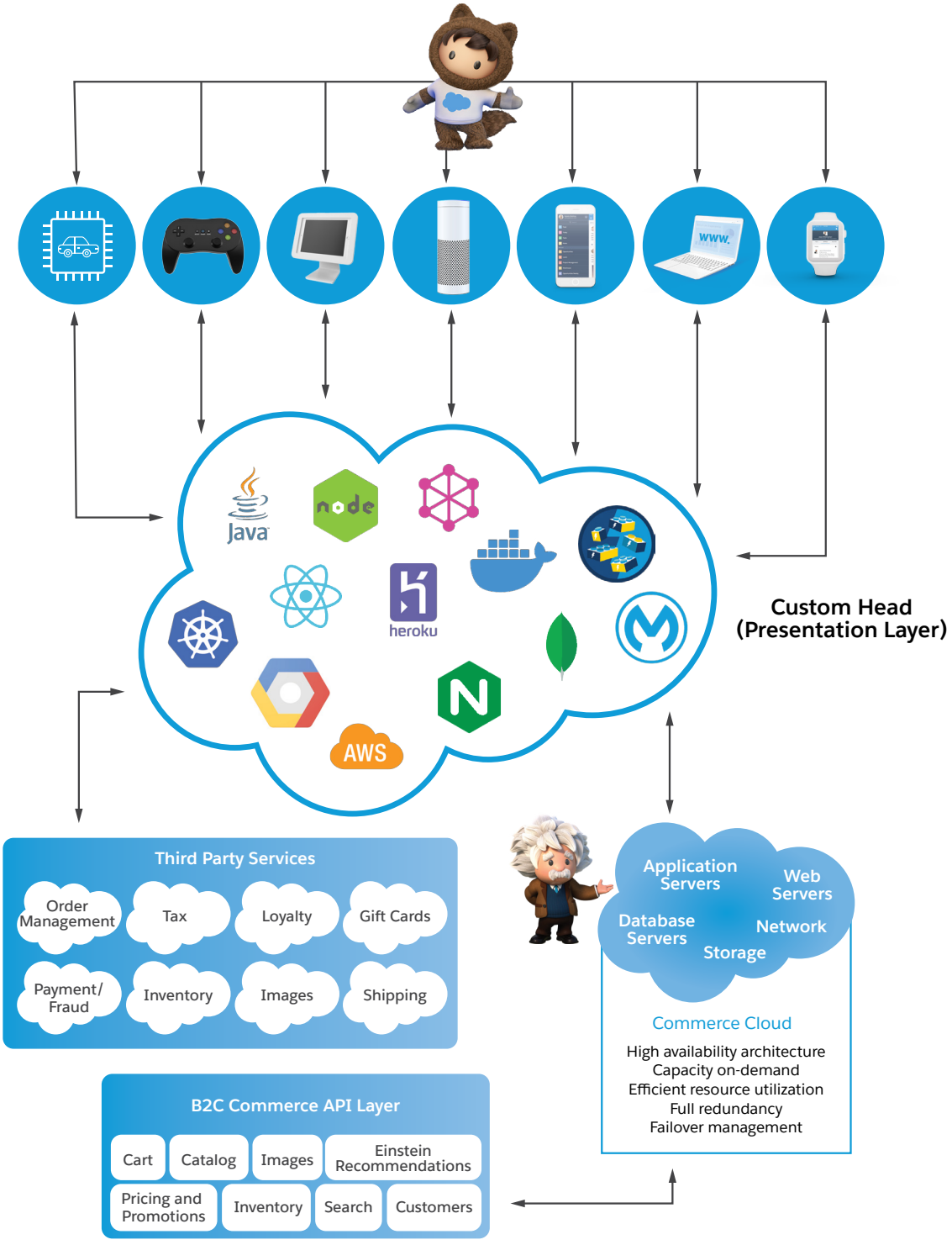
By now you should have a clear understanding of what it takes to develop a headless solution that integrates with Salesforce B2C Commerce. While developing a headless solution is a large undertaking that requires extremely knowledgeable development, support, and infrastructure teams, going headless gives you the flexibility to provide the functionality you need in your front end when you need it.

> *The Salesforce B2C Commerce platform provides flexible and highly performant APIs for securely accessing our back end, allowing you to focus on your front-end user experience.*

If you do not intend to build and maintain your headless solution entirely in-house, our experienced partner network, in conjunction with Salesforce Client Services, is available to assist with implementations.

5 Appendix

We've covered what a headless architecture looks like at a high level. However, no one architecture suits every need.



Refer to the table below for an overview of some of the industry- standard technologies recommended for use in developing and managing your custom head. While some of these technologies overlap when used in conjunction with one another, the table can serve as a starting point for evaluating technologies to suit your needs.

Technology	Description	Information	Tutorial	Category
Lightning Web Components	Salesforce Web UI Framework providing custom UI elements built using HDML and modern JaveScript	lwc.dev	Quick Start: Lightning Web Components on Trailhead	Front End
NodeJS	JavaScript runtime built on Chrome's V8 VM JavaScript engine	nodejs.org	w3schools.com	Front End
React	A component-based JavaScript library for building user interfaces	reactjs.org	Intro to React	Front End
Java	Application Development language for back end systems	oracle.com/OpenJDK	The Java Tutorials	Back End
GraphQL	A query language for designing API's and server-side runtime for executing queries using custom type system	graphql.org	Introduction to GraphQL	Back End
Heroku	Salesforce cloud Platform as a Service (PaaS) allowing you to build, deliver, monitor, and scale your custom applications	heroku.com	Getting Started on Heroku	Container Management
Kubernetes	Open Source container orchestration system for automating application deployment, scaling, and management	kubernetes.io	Tutorials	Container Management
Docker	Enterprise container runtime	docker.com	Get Started with Docker	Container Management
Amazon Web Services (AWS)	Amazon Cloud PaaS	aws.amazon.com	10-Minute Tutorials	Cloud Services
Google Cloud Platform (GCP)	Google Cloud PaaS	cloud.google.com	Getting Started with Google Cloud Platform	Cloud Services
NGINX	High performance load balancer, web server, and reverse proxy	nginx.com	Getting Started	Web Tier
MongoDB	Document-based (NoSQL) distributed database	mongodb.com	Getting Started	Storage
Mulesoft	Design high performance API's and integrations with eCommerce platforms using pre-built connectors and integration templates	mulesoft.com	Tutorials	API Gateway
Redis	In-memory, distributed data structure database	redis.io	Interactive Tutorial	Cache