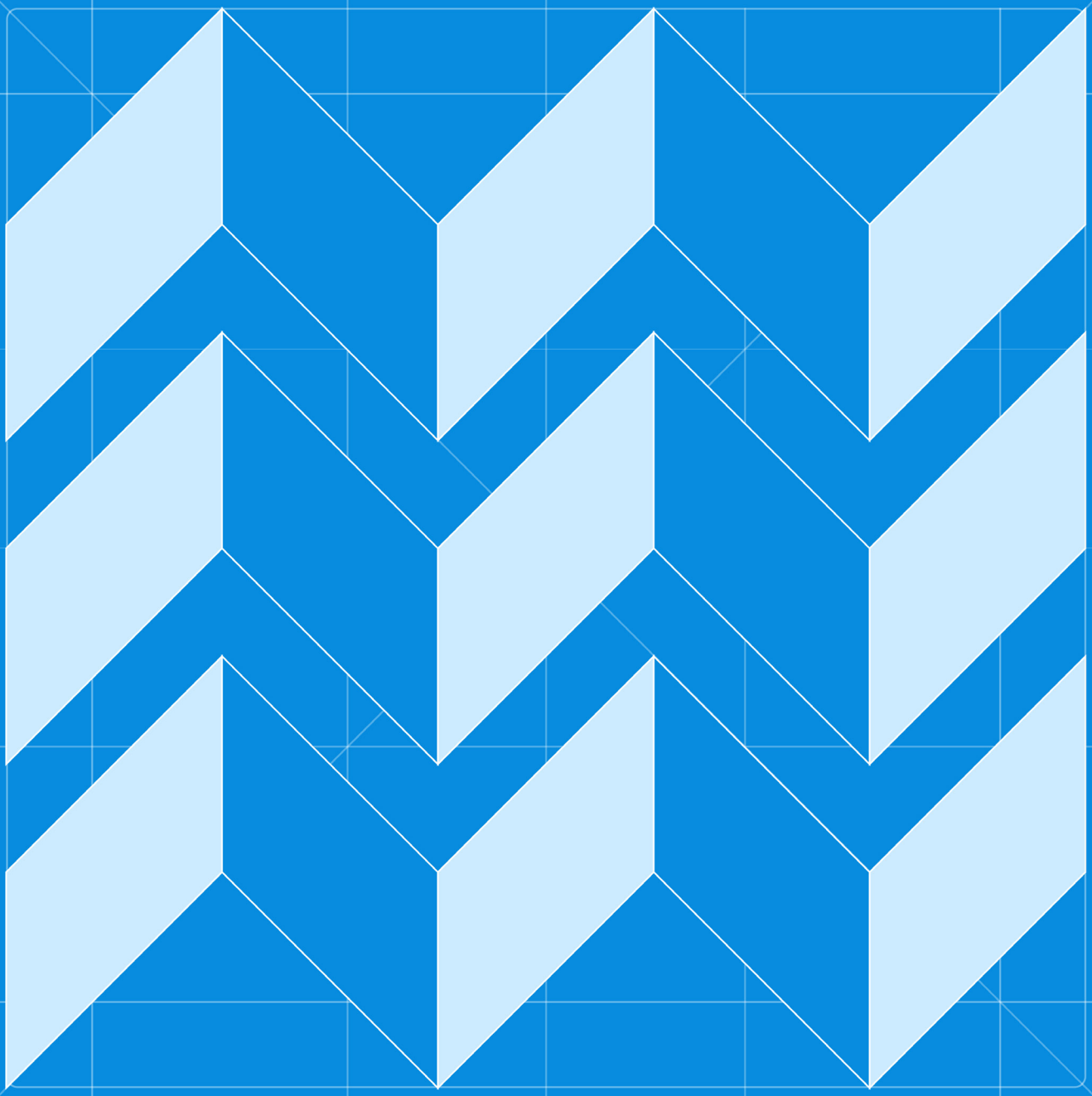
commercetools
Next generation commerce



Blueprint for a Modern Commerce Architecture™

An overview of templates for deploying commercetools

www.commercetools.com

Table of Contents

Executive Summary

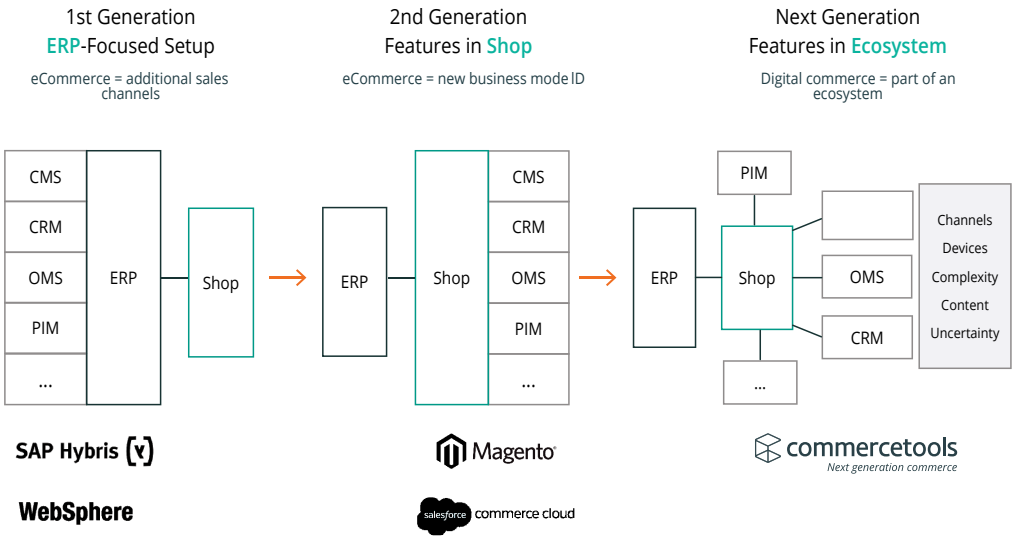
Customers now expect commerce to come to them across many different touchpoints. Combined with the increasing pressure to reduce operational costs while requiring constant innovation, retailers are being forced to rapidly modernize their commerce architecture. Not since eCommerce first became imperative during the 1990s have enterprises required such a monumental shift in the way they engage, deliver content across channels, sell and fulfill. Customer-facing sales channels have grown from the standard storefront and desktop webstore to contact points on mobile, television and streaming media, social media, and IoT devices.

The large monolithic applications that were the commerce platform solution of choice in the 1990s have become costly, burdensome, and ineffective at rapidly adapting to consumer needs. Modern commerce requires the capabilities of the cloud and the flexibility of microservices to rapidly anticipate and respond to consumer behavior. Only cloud-native, high-speed API, microservice, and event-based architectures allow for rapid application development with unified real-time data as required by consumer-centric applications.

While enterprises realize the need, they often have limited budgets and limited knowledge of how to transform their architectures to **MACH**. commercetools is meeting this need with the leading **M**icroservices-backed, **A**PI-first, **C**loud-native, **H**eadless commerce solution and a publicly available open-source Blueprint Architecture that allows enterprises to rapidly transform to a modern commerce architecture without starting from zero.

In this paper, we will cover:

- How monolithic applications impact the enterprise
- How to overcome conflicting objectives with the Blueprint Architecture
- How to use the commercetools Blueprint to create your own **MACH** commerce architecture rapidly



Blueprint for a Modern Commerce Architecture

An overview of the templates for deploying the commercetools API-based commerce platform

Business Issues

Once the organization has determined that an API-first, microservice-backed, cloud-native commerce architecture is the right solution, the inevitable next question is "now what?". The organization may have some existing applications with APIs built on top, or perhaps small teams have even built a few one-off microservices exposed as APIs, but there is no institutional knowledge around what or how to transform the organization.

The existing enterprise architecture is typically so complex and interdependent that the business suffers paralysis regarding where and how to start. Modern cloud-centric organizational transformation typically has four tracks (see figure 1) that are simultaneously vying for dollars and priority. All need to be in place to recognize the full transformative value, but a big-bang approach is both reckless and costly. Priority needs to be given to those capabilities that are the most consumer and cost-focused — thereby meeting both the revenue and cost reduction goals of the organization while presenting the highest ROI.

Transformation is typically centered around the following four capability tracks:

- Cloud and Digital Transformation
- CI/CD-End-to-End Ownership
- Microservices/ API/ Event Architecture
- Omnichannel Consumer & Corporate

Cloud Transformation

There is more to "going to the cloud" than simply copying your footprint into someone else's data center. Cloud transformation is a shift in technical operations, infrastructure costing, and most importantly, the way applications work. The cloud, in turn, becomes the fabric on which the digital commerce platform is built, operated, and grown. The simple "lift and shift" of an application does little to gain the value of a cloud strategy.

When utilizing a cloud-based architecture, organizations are looking for the following capabilities from applications:

1. **DevOps:** The IT organization moves to a culture and practice of unifying software development (Dev) and software operations (Ops). The goal being shorter development cycles, more frequent deployments, and more dependable releases, in close alignment with business objectives. This is done with a focus on automation and monitoring throughout the development, testing, deployment, and operations stages.
2. **Fully Elastic Consumption:** Scaling up and down, in an automated fashion, based on machine-based monitoring to maintain optimal performance at the lowest possible cost is typically the biggest benefit of a cloud transformation. The infrastructure scales up and down based on usage — even to the point of only existing when required. The need to estimate server/network usage a year in advance so hardware can be budgeted, ordered, and deployed becomes a thing of the past. A virtually unlimited set of hardware is available the instant it is needed — and discarded the moment it is under-utilized.
3. **Geo-redundancy:** Duplicating data centers (DCs) and applications based on concerns of a statistically unlikely occurrence is an expensive form of insurance. Even when we optimize the cost spent by running live out of both data centers and sizing so that normal traffic could be entirely run from one DC in the event of a disaster, it means our insurance policy only covers us when it matters least. It is not peak, Black Friday or Cyber Monday. When 70% of sales happen in two months, true geo-redundancy in the event of a disaster needs to be in place. In an active/passive model, cloud-based geo-redundancy allows your off-DC to be turned off with near-zero cost while your live-DCs are operating and scaling as designed. Cloud-native models typically run active/active as the need for a passive environment is typically redundant or unnecessary due to auto-scaling capabilities.
4. **Rapid, low-cost development:** Spinning up new or modified applications require minimal investment as the underlying infrastructure is on-demand. Additionally, public cloud offered SaaS and PaaS solutions allow for focusing on coding applications or configuring for a business need instead of focusing on deployment, scaling, hardware, and data storage. If the application is not successful, it can be quickly removed with minimal sunk-costs.

Continuous Integration/Continuous Deployment: End-to-End Ownership

Minimizing the time from ideation to launch is an ongoing quest in technology. With the advent of automation technologies around version control, testing, and deployment, the speed of change can be nearly instantaneous with multiple deploys per day. The only limitation is how fast change can be coded. Fixes and updates go from version control to production in a matter of minutes – vs. the weeks, months, or years we face with monolithic applications.

The way code is deployed also changes in the CI/CD world. Three examples are:

1. **Blue Green Deploys** - two identical environments where green hosts the present production version while blue hosts the upgraded environment. You run smoke

tests or any other tests in the blue environment while you prime the operating system, cache, etc. When things are fully running and ready, you point the load balancer/router to the blue environment, which now becomes the production environment.

2. **A/B Testing** - allows the enterprise to test features in the application for things like usability, popularity, noticeability, etc. If a feature presents value, it is rolled out to the remainder of the platform.
3. **Canary Releases** - send out a version of the application to see how it performs, integrate with other applications, utilize memory, etc. The idea here is that no matter how much we test in lower environments, issues always arise in production. This way, the deployment issues are isolated and can be rolled back before broad release.

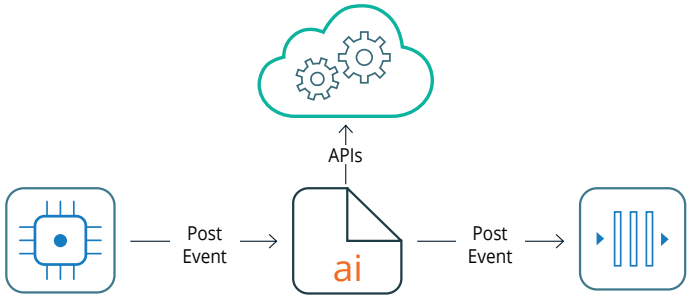
While the tooling around building development pipelines is part of the process, it is only a part of what is required. Using a scaled agile development approach to replace the waterfall methodology, required by monolithic applications, is essential as the process matches the organizational structure.

Instead of organizing teams around skill sets; such as DBAs, middle layer coders, frontend developers, operations, etc., development is organized around a functionality set, and teams become vertical in nature. All skills required around a given functionality are available within the team. For more information on this topic, I recommend reading here on how Spotify has implemented a scaled agile development process.

Microservice/API/Event-Based Architecture

In an ideal world, all data and all processes would be available real-time to any valid application. This is the idea behind the API/Event economy. Simply put, the idea is that any data or processing is made available the instant it is required. An API allows you to request or post data to a process. Events, in turn, notify consuming entities that changes have occurred within the system.

An order is a simple example of how APIs and events work together. An order may be submitted via an API call while changes to the order state, such as received or shipped, would be posted as events. Events become particularly interesting as IoT-connected devices and artificial intelligence-driven actions increase in usage.



IoT event-based AI mechanism. The AI mechanism reacts to the incoming event by posting its own event and/or exposing as an API.

Why Monoliths with APIs Added on Do Not Work

In the days of the monolith, all data was stored in one massive database with a middle tier and view tier built on top. This pattern was duplicated for different business purposes over and over within the enterprise. Any application that was going to rely on the monolith's data would either have to be a part of the monolith or would need to store a duplicate version of the data. Batch jobs and ETL (Extraction, Transformation, and Loading) became the wiring that held together corporate architectures.

As needs changed, software vendors added layers on top of the monolith to expose APIs for use by external applications. This was meant to modify ETL processes to be less batch in nature, but was never intended to fully support the high-speed scale required for omnichannel commerce. Because the API was an afterthought of the monolith, it was never meant to be a truly high-speed entity. Typically, any large load on an API causes the internal processing of the monolith to slow or fail due to the unexpected processing burden. This meant duplicating data in caching and content delivery layers in an attempt to cover up the weaknesses of the system.

The modern commerce platform has highly available, highly scalable APIs exposed to multiple consumer endpoints.

Omnichannel Consumer and Corporate Applications

In the original retail model, there were two primary touchpoints: storefront and catalog. eCommerce came along in the 1990s, and the catalog was largely replaced with the desktop webstore. Thus, organizations typically had one set of operations for stores and one for eCommerce. The software that supported this dual front was logically monolithic. eCommerce was treated as just another store in many instances and fed the data it needed.

Customer touchpoints are everywhere now. While companies are trying to catch up by building mobile apps, innovative companies like Disney® are unifying customer experiences online and in parks with their Magic Band, while Amazon uses RFID technology to eliminate cash registers at their experimental physical stores. Touchpoints like these are created by building small apps on top of the API economy.

The objective for every retailer should be to allow consumers to easily make a purchase everywhere they interact with the brand. The monolithic nature of the old eCommerce platforms just cannot be modified to handle dozens of simultaneous applications (both internal and external) all vying for the same instantaneous real-time updated data. They were built in an era of web-only purchasing, not the omnichannel experiences of the modern shopper.

Omnichannel Delivery and the Customer Experience

The omnichannel experience of modern shoppers has expanded far beyond the device. Smart spaces, augmented and virtual realities, and the always-connected customer are blurring the lines between digital and traditional shopping. Resources may be divided by channel behind the scenes, but when your customer is standing in a physical store, looking up a discount code in their email, and using the in-store kiosk to see if their item is in stock, they expect the entire shopping experience to work together seamlessly.

The 2019 Salesforce report, "The State of the Connected Customer," found that 64% of customers are using different devices to start and complete transactions and 84% stated that the experience a company provides is as important as its products and services. No other industry is under as much pressure to ensure their omnichannel efforts don't just perform, but delight.

This is the role content plays in commerce. Good content turns a digital product catalogue into an experience. An AR app that shows what that couch would look like in your apartment needs great pictures. Making service appointments from within your connected car requires a quality script for voice guidance. Modern customer support requires helpful content that is available via web, app, chat, voice, and within a smart product.

A content model that meets the demands of the omnichannel shopper should be aligned with the Blueprint Architecture for Microservices Commerce. Many monolithic content systems still create duplicate content for each channel and require complex, custom integrations to get that content to link to commerce information. Modern, microservices-based content systems provide a structure that makes it simple to pivot to new channels and use critical data from your commerce, CRM, or other systems to drive dynamic delivery.

For example, a top fashion-forward apparel retailer in the United States had been hard coding the graphics in their customer-facing app. Not being linked to commerce information, the graphics for sold out items could take up prime app real estate for weeks until manually removed.

As the holiday season was approaching, the retailer was eight months into a nine-month deployment of a monolithic solution for digital commerce — and still four months behind schedule. Not wanting another Black Friday with an app that presented shoppers with out of stock items, the company decided to make a switch to the commercetools **MACH** solution. By putting a microservice design at the core of both commerce and content, they were able to launch their new, dynamic app experience a full two weeks before their target date. Having created this foundation of microservices commerce, the retailer is now expanding the experience seamlessly to interactive, in-store displays.

Solution: Create Your Own Cloud-Native Architecture

Innovative organizations want to combine the building of a sustainable, modern architecture with the highest impact customer-facing capabilities. The commercetools Blueprint for Modern Commerce Architecture™ gives you a foundation on which you can build and grow your digital platform.

The Blueprint is designed for focuses on a cloud-native, microservices-based architecture with API and event-based transaction capabilities. The architecture is useful for interacting with both commercetools and other API-based providers, as well as connecting the internal ERP and other legacy systems.

At its core, the Blueprint is a cloud-native foundation on which to build the omnichannel commerce capability that enterprises seek. The Blueprint is meant to be deployed into the public cloud. The enterprise is the owner of the deployment, all the code, and all the networking. It is a fully stand-alone set of infrastructure and tools separate from commercetools and other vendors.

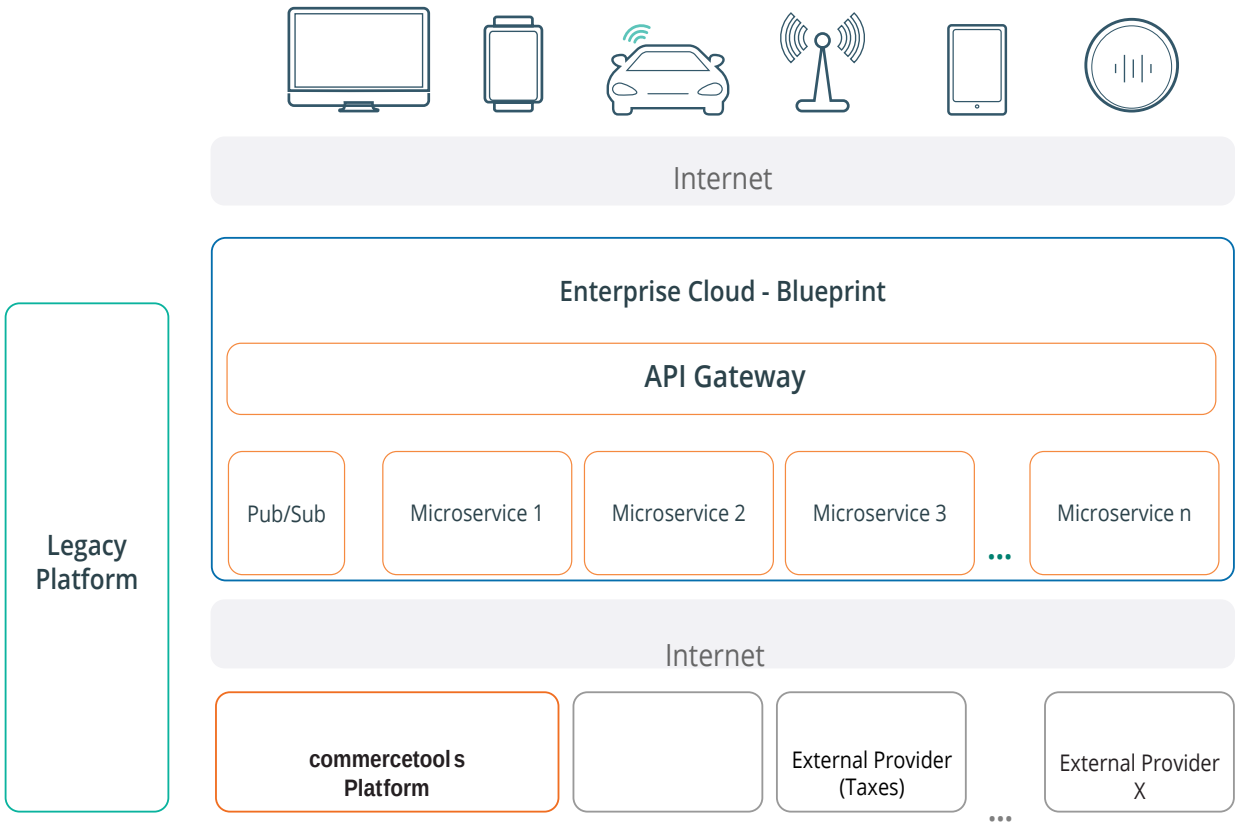
The Blueprint creates a level of abstraction between the corporate environment and providers. The enterprise then can define its own API contracts with multiple consuming applications. If the supporting component behind the API changes, consuming applications are unaffected and do not require code updates or testing. This means that providers can be replaced based on the value of their APIs vs. competitors in the market. By introducing the Blueprint abstraction, the enterprise does not tie its consuming applications directly to any licensed provider. The enterprise retains control of its public interface.

Proven Methodology to Rapidly Build, Test and Deploy

The commercetools Blueprint for Modern Commerce Architecture™ allows companies implementing the commercetools platform to rapidly build, test, and deploy a cloud-native, microservice-based architecture just as commercetools uses, in a rapid, reproducible fashion into their cloud environment.

The Blueprint creates a functional abstraction API, event and service layer with full CI/CD and auto-scaling capabilities that are fully controlled by the enterprise. The commercetools APIs and events are abstracted away from direct consuming applications behind a corporate infrastructure.

Unlike typical blueprints or open-source samples, the commercetools Blueprints are full architectures that can be workable in a very short time period. If this were a lawn, the commercetools Blueprint would be the sod – not just seed. While there is an effort to deploy and connect, there is a full set of workable services, development pipelines, and connections ready to go.



Vendor software is obfuscated behind enterprise-owned microservices and APIs. Full control of the API interface rests with the enterprise. All consuming applications engage with enterprise APIs, thereby allowing downstream vendors to be replaced or upgraded without any consuming application impact.

What Is in the Blueprint?

There are multiple blueprints available. Each has a mono-repository and instructions on how to deploy a fully functional blueprint. In addition, individual pieces of the Blueprint can be utilized if the full architecture is not required. Examples of this would be the CI/CD pipelines, sample microservices (both serverless and standing service in nature).

The Blueprint addresses the transformative requirements enterprises seek in the following ways:

Cloud Transformation: The Blueprint is a cloud-native, auto-scaling architecture that is API and event-based. While the core Blueprint microservices integration is with commercetools, the microservices can be cloned or passed through to connect easily to other API vendors. Ultimately, the architecture provides the connection functionality to commercetools along with the pattern of integration for other API based vendors. There are two architectures available:

- Blueprint GCP PaaS – Organizations that want to push as much of the operational activities as possible while still owning the code can utilize a blueprint built entirely on the Google Cloud PaaS offerings. This means that all control of server-level functionality and auto-scaling gets pushed to become Google's responsibility. The enterprise need only provide the coding, testing, and deploy of functionality. Google handles the rest.
- Blueprint K8 – A cloud-independent Blueprint with fully containerized code built within a Kubernetes deployment. This means that AWS, GCP, or Azure can all be used. Each component is fully independent, and multiple clouds can be used simultaneously. Ultimately, this means more control rests in the hands of the enterprise.

CI/CD – End-to-End Ownership: A full continuous integration/continuous delivery pipeline, including Github-based source repository, automated testing scripts, and deployment built on Kubernetes.

Event-Based Architecture: The Blueprint is a microservice-based architecture with exposure via APIs and events. Connections to the commercetools set of APIs and events are built into each of the representative services.

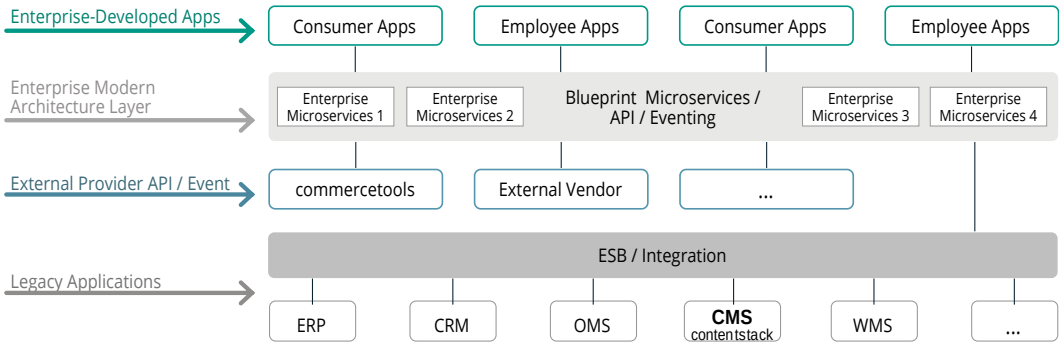
Omnichannel Consumer & Corporate: While no consuming applications are built as part of the architecture, a full high-speed corporate API/Eventing layer is available to access via the include API gateway. This allows for the rapid development of both corporate facing and consumer-facing applications on mobile, IoT, desktop, etc.

Architectural Strategy of the Blueprint

In an ideal world, the commercetools Blueprint Architecture would be an end-to-end enterprise architecture that could do everything any enterprise needed in a high-speed, cloud-native, microservice-based API fashion. Replacing ERP, warehousing, and innumerable other large monoliths with a one-time project is an unsound practice. Thus, the Blueprint focuses on the consumer-facing, commerce layer of the architecture and not on the operational side.

Inherent in deploying the Blueprint is that the underlying legacy architecture core is not modified. This strategy is often referred to as Dual-Speed or Hybrid architecture. The idea is that the consumer-facing technologies and data are built on very high speed – scalable, microservices architecture, while the core infrastructure of the organization is not modified (at least in the initial phases).

This allows the organization to compete and lead in consumer-facing applications, while the core of the architecture can be modified over time in a reasonable and methodical manner. Other components of the enterprise can be removed or reduced through a strangle pattern, while the outward face of the organization is cutting-edge.



Utilize the Blueprint to create a high-speed commerce face while removing any tight vendor coupling and legacy system dependency.

The Blueprint layer is owned and managed by the enterprise. The endpoints it is exposing to consuming apps are then always controlled by the enterprise. If, for example, one of the commercetools microservices is replaced by a microservice built in-house, the consuming app exposed endpoint would not need to change. Additionally, the Blueprint layer offers another place where customization can occur. As the enterprise defines its own API interface, the enterprise microservice layer defined by the Blueprint Architecture contains enterprise-built custom microservices, transforms, customization on vendor APIs, GraphQL capabilities, and multi-API consolidation.

As with any Architectural structure, the enterprise should determine if the Blueprint architecture has more benefits versus directly integrating with API vendors such as commercetools. Both have an element of dual-speed in the architecture, but the Blueprint centralizes control of the enterprise API set into something wholly owned and controlled by the enterprise. See the comparison chart below.

Benefits of Enterprise-Owned Service Layer Model

- Full control of the app-facing layer
- Consumers applications do not change when functionality vendors are replaced
- Additional point of customization
- Optimize APIs for consuming applications
- Organizational change starting point
- Rapid development environment
- Automation built-in

Benefits of Directly Connecting Consuming Applications to API Vendors

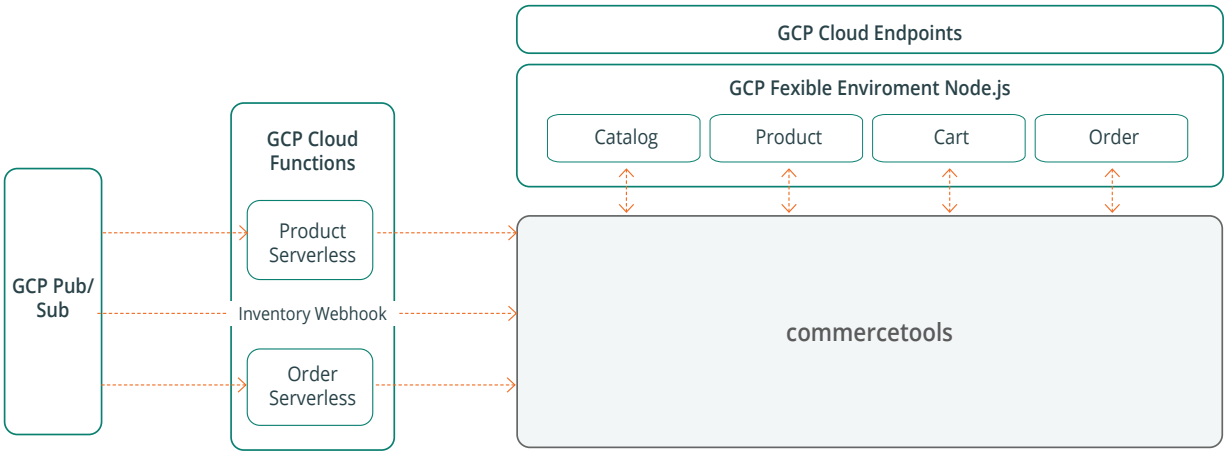
- No additional hop - transactionally, this is going to be in the 10 to 50-millisecond range
- No additional monitoring and issue diagnosis layer
- No additional cloud environment costs
- No additional development cost
- Automation built-in

Blueprint Offerings

Two Blueprint Architectures are available. Blueprint GCP PaaS is meant for organizations who want to move most of the operations side of their cloud offerings to the cloud provider. Blueprint K8s has significantly more flexibility but requires the organization to take more control over what, where, and how they deploy their architecture.

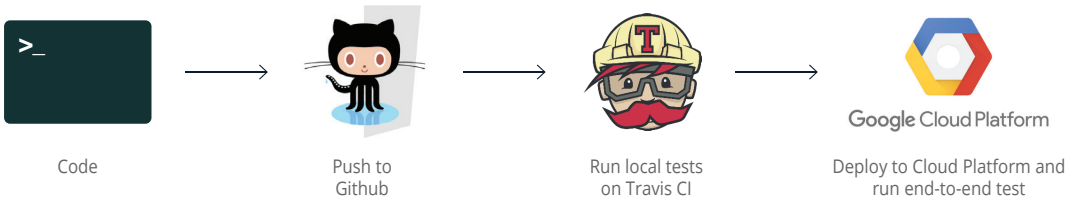
Blueprint GCP PaaS

Blueprint GCP PaaS is dependent on the use of Google Cloud Platform. It is an entirely Node.js based set of code. It utilizes the GCP's set of SaaS and PaaS offerings to let Google handle things like scaling and machine sizing. There is almost no required knowledge of hardware required in this Blueprint. It is entirely a coding exercise and is meant to minimize the operational knowledge and resource requirements for deployment.



Components of Blueprint GCP PaaS

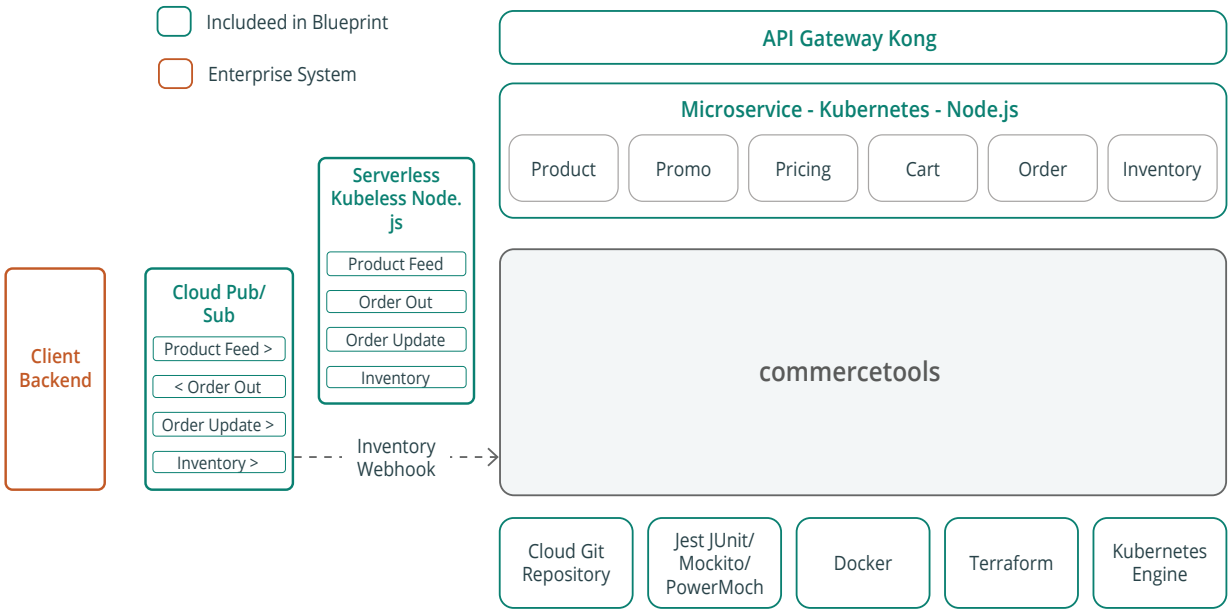
- API Gateway: Cloud Endpoints - Google Cloud Platforms API gateway offering
- Serverless Microservice: Google Cloud Function - Node.js serverless function within the Google ecosystem
- Microservice: Google Flexible Environment - Node.js implementation of “always-on” microservices
- Cloud Pub/Sub: Google Cloud Pub/Sub - Eventing environment
- Monitoring: Stackdriver
- CI/CD Pipeline: Pictured below, includes automated testing with Jest and deploy



Blueprint K8s

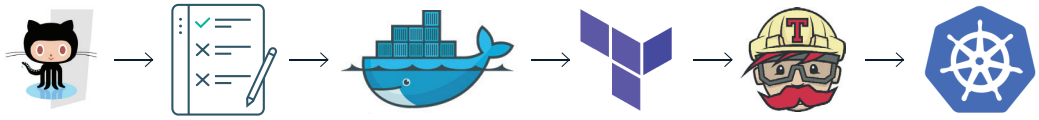
Blueprint K8s has no cloud provider based dependencies. All functionality is containerized via docker, as well as deployed and scaled via Kubernetes. This does mean that the cloud to which it is deployed must support Kubernetes; of which AWS, GCP, and Azure all support. Where Blueprint GCP PaaS relied on Google Cloud autoscaling and networking, Blueprint K8s gives full control back to the enterprise.

The Blueprint provides a pattern and deployment for the enterprise to move to a large scale cloud architecture in a matter of days.



Components of Blueprint K8s Implementation

- API Gateway: Kong (https://konghq.com/)
- Serverless Microservice: Kubernetes native serverless offering called Kubeless. Written in Node.js for the included code, but does not have a language restriction like GCP or any of the cloud providers serverless offering out today.
- Microservice: Kubernetes based deploy of a Node.js and Java-based “always-on” microservices.
- Cloud Pub/Sub: Kubeless based offering written in Node.js
- Logging and Monitoring are left for the implementation. Every enterprise has tools in place, and there is no need to force a second set as part of the implementation. Each cloud has logging by default as part of the deploy, which can be integrated or replaced.
- CI/CD Pipeline: Pictured below, includes automated testing in Jest or JUnit (Mockito/ PowerMock libraries), Terraform and Kubernetes deploy.



Operational Components

Because Blueprint K8s can be deployed into any of the Kubernetes-supporting clouds (AWS, GCP, or Azure), the cloud-dependent operational and management functions are left to the implementation.

Getting Started

The goal behind the Blueprints is to accelerate and define many of the early steps in an implementation. Organizations typically know the set of functionalities they need to build in a project. The Blueprint gives the development team a template from which to start. It defines everything from the directory structure to deployment pipelines, as well as containing coded solutions and template microservices to assist the team in jumping over the “getting started hurdles”. The goal is to save a typical project six to eight weeks and allow the team to deliver functionality quickly.

Utilizing either of the Blueprints as an accelerator for your platform can be done in many ways. The most obvious use is as part of a digital transformation project as you move to headless commerce built around commercetools. Enterprises who wish to slowly modify their platform because of business or technical reasoning would likely utilize the architecture to build out a greenfield site or replace portions of their digital experience piece by piece over time. If you have already started the journey, it is likely that you would only use components such as the CI/CD pipeline and a few of the sample services to accelerate a commercetools implementation.

Conclusion

In an ideal world, the commercetools Blueprint Architecture would be an end-to-end enterprise architecture that could do everything any enterprise needed in a high-speed, cloud-native, microservice-based API fashion. Replacing ERP, warehousing, and innumerable other large monoliths with a one-time project is an unsound practice. Thus, the blueprint focuses on the consumer-facing, commerce layer of the architecture and not on the operational side.

Inherent in deploying the Blueprint is that the underlying legacy architecture core is not modified. This strategy is often referred to as Dual-Speed or Hybrid architecture. The idea is that the consumer-facing technologies and data are built on very high speed – scalable, microservices architecture while the core, infrastructure of the organization is not modified (at least in the initial phases).

This allows the organization to compete and lead in consumer-facing applications while the core of the architecture can be modified over time in a reasonable and methodical manner. Other components of the enterprise can be removed or reduced through a strangle pattern, while the outward face of the organization is cutting-edge. The Blueprint layer is owned and managed by the enterprise. The endpoints it is exposing to consuming apps are then always controlled by the enterprise. If, for example, one of the commercetools microservices is replaced by a microservice built in-house, the consuming app exposed endpoint would not need to change.

Additionally, the Blueprint layer offers another place where customization can occur. As the enterprise defines its own API interface, the enterprise microservice layer defined by the Blueprint Architecture contains enterprise-built custom microservices, transforms, customization on vendor APIs, GraphQL capabilities and multi-API consolidation.

As with any architectural structure, the enterprise should determine if the Blueprint Architecture has more benefits versus directly integrating with API vendors such as commercetools. Both have an element of Dual-Speed in the architecture, but the Blueprint centralizes control of the enterprise API set into something wholly owned and controlled by the enterprise.

About commercetools

commercetools is a next-generation software technology company that offers a true cloud commerce platform, providing the building blocks for the new digital commerce age. Our leading-edge API approach helps retailers create brand value by empowering commerce teams to design unique and engaging digital commerce experiences everywhere – today and in the future. Our agile, componentized architecture improves profitability by significantly reducing development time and resources required to migrate to modern commerce technology to meet new customer demands.

The innovative platform design enables commerce possibilities for the future by offering the option to either use the platform's entire set of features or deploy individual services, á la carte over time. This state-of-the-art architecture is the perfect starting point for customized microservices, enabling retailers to significantly reduce time-to-market for innovative commerce functionalities.

With offices in Germany and the United States, as well as presence across general Europe and Asia Pacific/Oceania, B2C and B2B companies from across the globe including well-known brands across many industries, including fashion, food and retail, trust commercetools to power their digital commerce business.

Contact Us

Europe - HQ
commercetools GmbH
Adams-Lehmann-Str. 44
80797 Munich, Germany
Tel. +49 (89) 99 82 996-0
info@commercetools.com

Americas
commercetools, Inc.
324 Blackwell, Suite 120
Durham, NC 27701
Tel. +1 212-220-3809
mail@commercetools.com

www.commercetools.com

Munich - Berlin - Jena - Amsterdam - London - Durham NC - Singapore - Melbourne