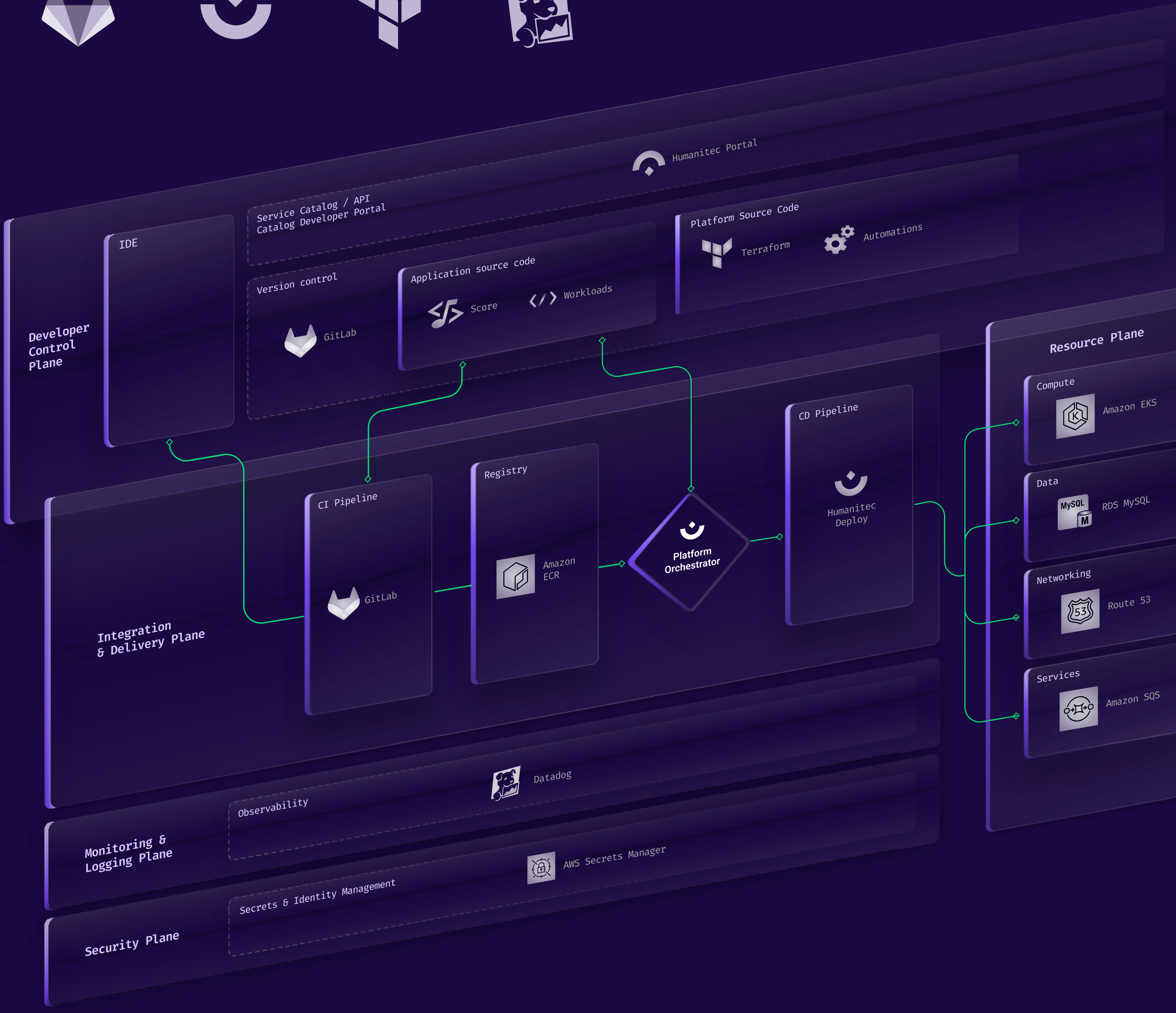
Reference architecture of an Internal Developer Platform on AWS

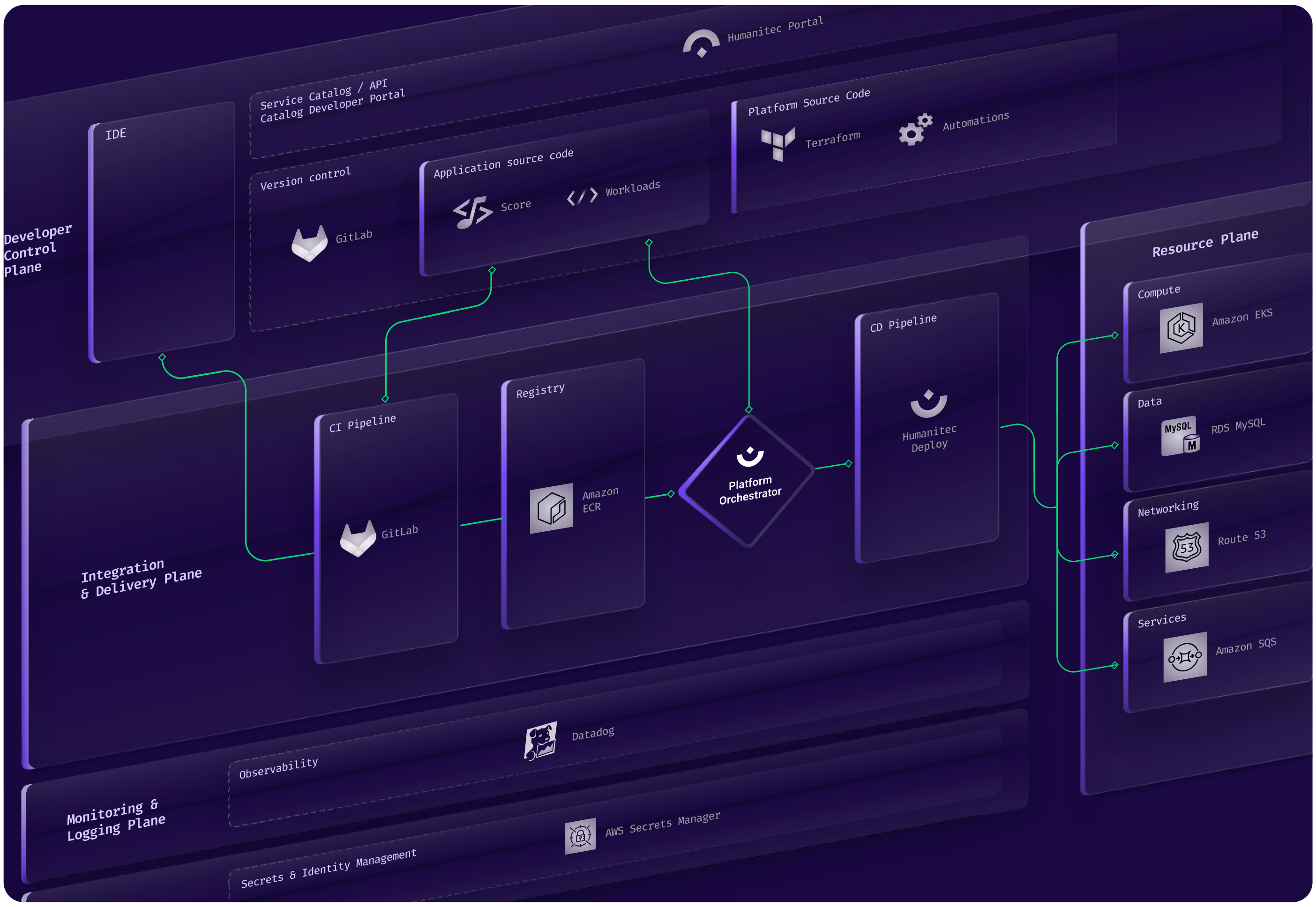


Platform Engineering

Introduction

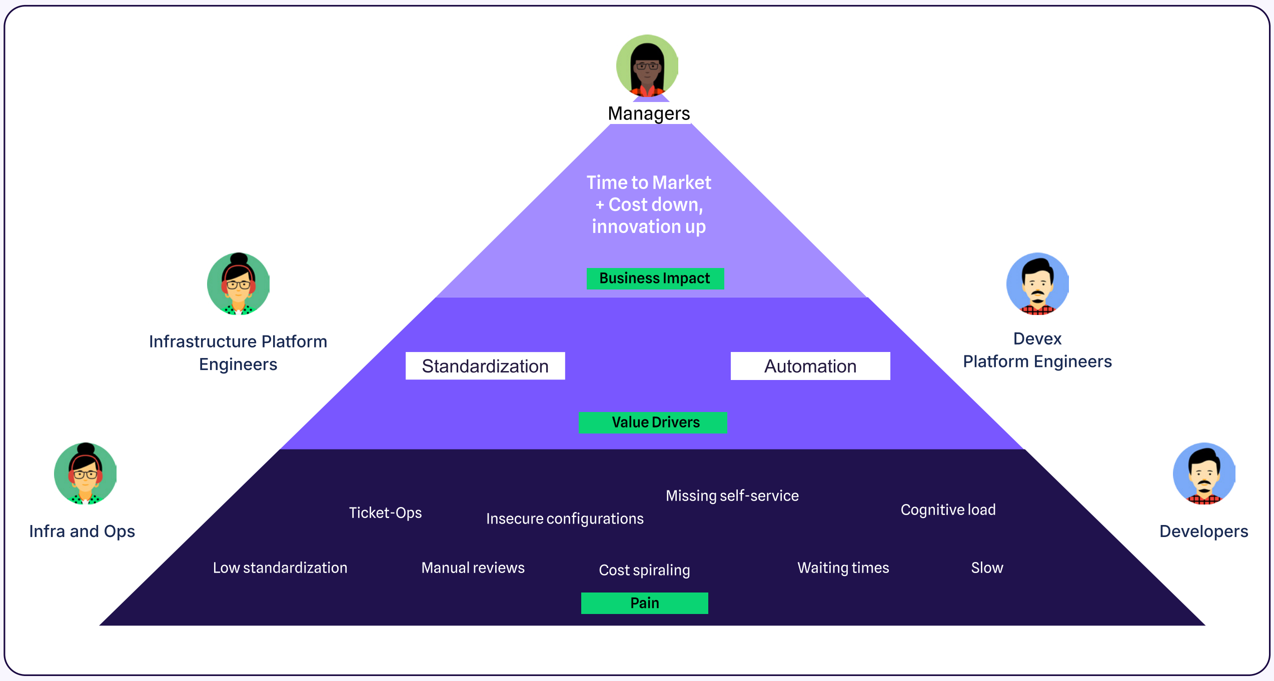
Organizations today must innovate quickly and reduce time to market in order to remain competitive. This requires transforming static CI/CD setups into modern enterprise-grade Internal Developer Platforms (IDPs) that improve developer productivity and increase Ops efficiency through standardization and automation. Well-designed IDPs eliminate ticket-based workflows and minimize the repetitive manual tasks that need to be performed under time constraints.

For developers, an IDP reduces cognitive load and DevOPs burnout and enables them to self-serve everything they need to be productive. Improving the developer experience leads to higher developer productivity, which ultimately results in a shorter time to market.



FROM PAIN TO VALUE - WHY YOUR PLATFORM MATTERS

Platform engineering addresses significant pain points experienced by both developers and infrastructure/operations teams, ultimately driving measurable business impact. By resolving these issues using the key principles and value drivers of platform engineering, organizations can achieve desired business outcomes like improved DevEx, cost savings, reduced churn or faster time to market.



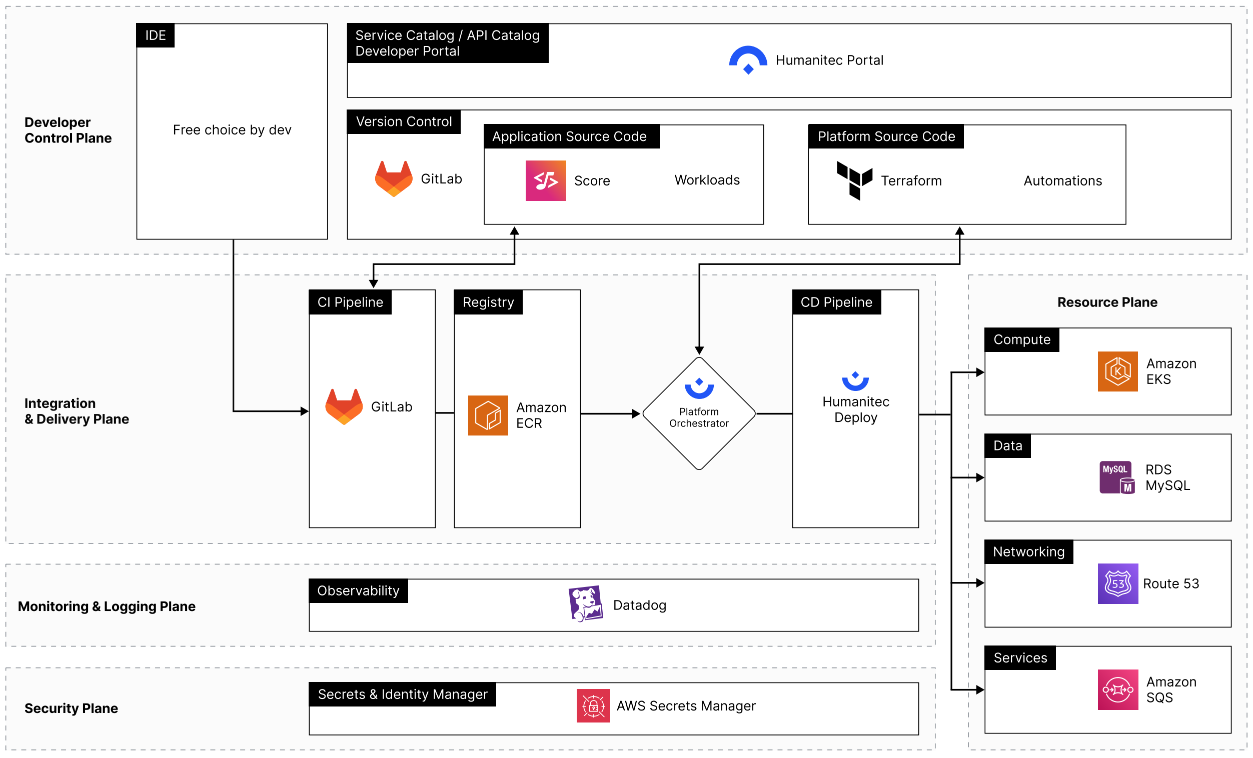
The question is, how do you build an enterprise-grade IDP, and where do you start? As an industry, we need to move beyond buzzwords and provide real-life examples of modern IDPs. While every platform looks different, certain common patterns emerge. To simplify matters, we have initiated a series of reference architecture whitepapers, where we walk through successful patterns in detail. These reference architectures represent platforms that have been implemented by community members and presented at Platform Engineering events or PlatformCon.

This whitepaper focuses on Convera's reference architecture of their Internal Developer Platform.

Convera's Internal Developer Platform

Convera, a company rooted in traditional finance, embarked on a significant transformation to become a leading-edge FinTech organization. This evolution was driven by the need to overcome several critical challenges, including a rigid and risky waterfall release model, the demands of a highly competitive market requiring modern technology, and the desire to be seen as a company that values innovation over stagnation. This transformation was presented by Igor Kantor at PlatformCon 2024, showcasing a remarkable blueprint for a Fintech enterprise. To meet these challenges head-on, Convera embraced platform engineering as a core strategy. This approach focused on three main pillars: streamlining development, empowering developers through autonomy, and ensuring operational efficiency.

REFERENCE ARCHITECTURE OF AN INTERNAL DEVELOPER PLATFORM ON AWS



The key to implementing this philosophy was the construction of a best-practice-focused Internal Developer Platform (IDP). The IDP, based on a well-defined reference architecture, aimed to create a smooth, uninterrupted flow of work, from the developer's laptop to revenue generation for customers. The architecture of this platform is detailed in the above diagram which shows off the different control planes. By moving away from manual processes such as "click ops" or "ticket ops," the platform sought to empower developers to deliver end-to-end value quickly and efficiently. Furthermore, the platform needed to operate with high efficiency and optimized costs.

"Platform engineering is truly about a fully automated digital supply chain and the humans that interact with it."

Igor Kantor
Director of Software Engineering at Convera

10 PROVEN BEST PRACTICES

Platform engineering best practices emphasize a product mindset when building an Internal Developer Platform (IDP). This approach is rooted in the understanding that a platform should be developed with the end-user, usually an application developer, but also other key stakeholders like security, I&O, architects and executives in mind. 10 best practices should be followed when developing an IDP.

01 Build vs. buy vs blend

The market reached a maturity level where it doesn't make sense anymore to build everything from scratch. You should make a business case and do your ROI calculations as early as possible. According to the 2023 study, the best-performing platform engineering teams blend OSS with commercial vendor offerings but do not build everything from scratch. Check the platform tooling landscape for inspiration.

02 Apply well established architecture patterns

With a three-tier architecture (presentation, application, data): start building from the backend; do not simply put a developer portal as a presentation layer on top of your existing setup and build additional logic into it. Build the house first, then the front door.

03 Everything as code

Consider code as the single source of truth which helps maintain transparency, increase reliability, and simplify maintenance. An IDP that's code-first at its core allows for disaster recovery, versioning, and structured product development principles. This does not exclude further interface offerings such as a UI (portal), CLI, or API.

04 Build golden paths over cages

Do not try to please everyone. To adapt to different situations, meet diverse needs, and benefit from evolving technologies, staying open-minded is key. But your platform design should not try to cover every technology on earth or convince every developer in a user base. Do not assume that you will be able to please 100% of developers. Instead, consider achieving 80% a great win.

05 Take an 80/20 attitude to platforming

Do not try to please everyone. To adapt to different situations, meet diverse needs, and benefit from evolving technologies, staying open-minded is key. But your platform design should not try to cover every technology on earth or convince every developer in a user base. Do not assume that you will be able to please 100% of developers. Instead, consider achieving 80% a great win.

06 Leave platform interface choice to the developer

To ensure adoption, give developers the freedom to use the interfaces they're most comfortable with and that best meet their needs. Provide the option to use an OSS workload specification like Score, a portal (GUI), CLI, or API.

07 Security from scratch

To get buy in, implement security best practices from the get-go. If the V1 of your platform doesn't fulfill security and compliance requirements and if there is no proof that the platform will even support ensuring security and compliance by design, security teams will veto and your platform initiative is dead before it could even properly start.

08 Measure from the beginning

Measure success with hard numbers to support informed decisions and generate stakeholder buy-in. Choose metrics wisely, considering both leading (e.g., automation and complexity scores) and lagging indicators (e.g., DORA metrics). Track leading indicators in non-production environments early on. Remember to include NPS scores for developer satisfaction, as well as stability metrics, SLOs, and SLAs.

09 Gain stakeholder buy-in

Make sure all stakeholders have a seat (besides the developers - your customers). From security to compliance and legal teams, from architects to I&O teams, and important for the funding of your platform engineering initiative: executives. Make sure you build a platform team where important stakeholders are represented by heralds and the team goals are aligned with those of your stakeholders.

10 Think about adoption from the first day

If the platform is not used, it is dead. This is about internal marketing/evangelism. Identify the right first team to onboard and make them advocates of your platform. They are essential for platform success and developer adoption.

Architectural components

The different areas of Convera's platform architecture are organized as planes that cluster certain functionalities. Let's zoom in on the different planes and see what technologies fulfill each function in each of them.

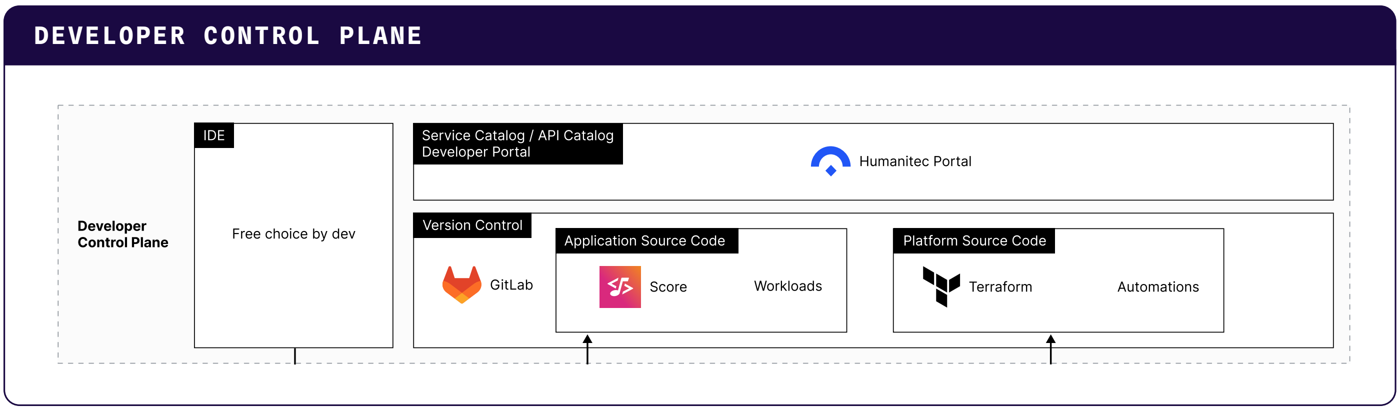
Developer Control Plane

Under the term Developer Control Plane, you find the primary "interfaces" developers can choose to use when interacting with the platform or applying any change. As discussed in the "proven best practices" section, organizations should leave interface changes to the developer on a workload-by-workload basis. We also recommend not breaking the developer's current workflow, which is why we default to code wherever possible.

Components used in this architecture

This plane is the platform users' primary configuration layer and interaction point. It harbors the following components:

- A Version Control System.VCS is a prominent example, but this can be any system that contains two types of repositories:
  - Application source code
  - Platform source code, e.g. using Terraform
- Workload specifications. The reference architecture uses Score.
- A portal for developers to interact with. This can be Backstage or any other portal on the market. This reference architecture uses the Humanitec Portal.



Following the best practice “everything as code” both the app and platform source code are stored in Git. The platform source code represents the configuration of the platform and is maintained using the IaC framework Terraform. Terraform is used for both managing the Humanitec Resource Definitions using the Humanitec Terraform provider, and for configuring the different automation systems. The primary interaction method for developers is designed to be code-driven using the Workload spec Score, to describe the Workload and dependent Resources in abstract terms. Git integrates with the IDE, the CI pipeline, and the portal using the GitLab API.

The portal layer offers a user interface on top of all platform capabilities that acts like a single pane of glass, including shortcuts to scaffolding new services, metrics, service catalogs, and additional self-service actions. The portal integrates with the VCS through its API, using plugins where available, and equally to the Platform Orchestrator. It might also pull additional data directly from the CI pipelines or project management systems like JIRA.

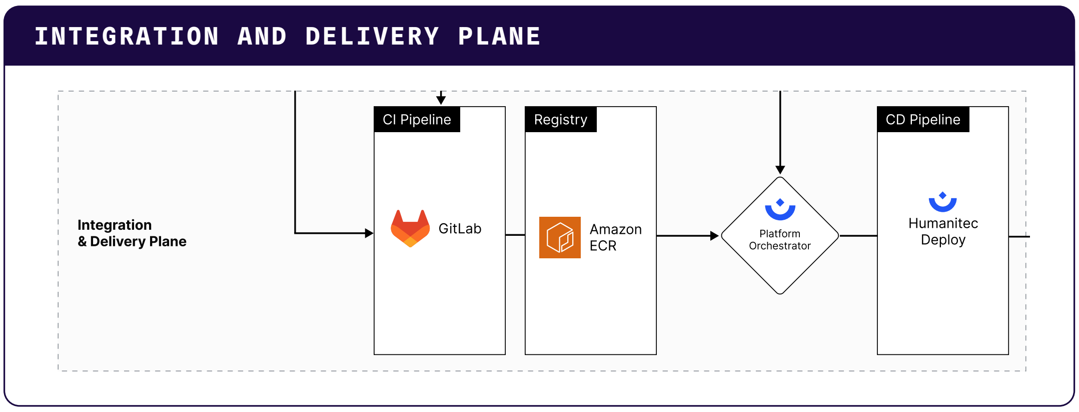
Integration and Delivery Plane

This plane is about building and storing the image, creating app and infra configs from the abstractions provided by the developers, and deploying the final state. It’s where the domains of developers and platform engineers meet.

Components used in this architecture

This plane usually contains four tools:

- A CI pipeline. This can be GitLab or any CI tooling on the market.
- The image registry holding your container images, in this case Amazon ECR.
- A Platform Orchestrator, which in our example is the Humanitec Platform Orchestrator.
- The CD system, which can be the Platform Orchestrator’s deployment capabilities as in this example, an external system triggered by the Platform Orchestrator using pipelines, or a setup in tandem with GitOps operators like Flux or ArgoCD.



Igor describes the deployment pipeline as a key part of their streamlined software development lifecycle (SDLC). The goal was to make the entire digital supply chain from a developer's laptop to revenue generation as cost-efficient, smooth, and easy to manage as possible.

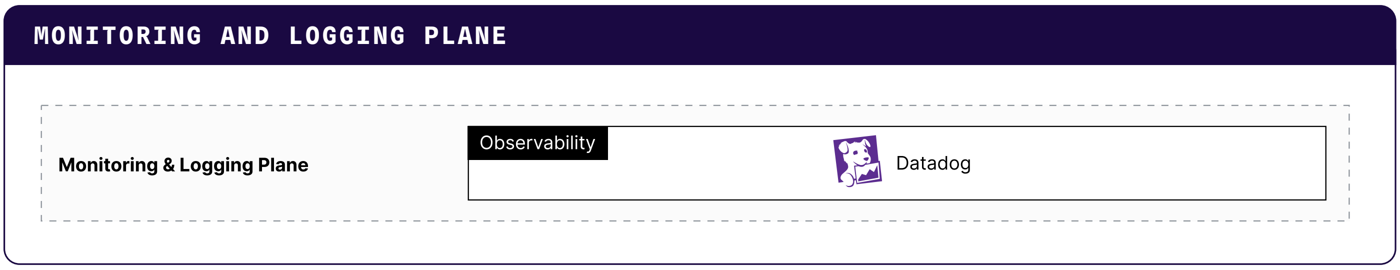
There are two key aspects to highlight here:

- **Developer self-service:** The aim is to enable developers to deliver end-to-end solutions without needing to interact with other teams (like DevOps or Cloud Engineering). By using Score (as part of the Developer Control Plane), developers can request the necessary resources themselves.
- **Platform Orchestration:** The Humanitec Platform Orchestrator is invoked to handle the actual deployment into the Kubernetes (EKS) cluster. The developers do not have to be involved with the actual deployment, which is fully automated. Based on Resource Definitions written in Terraform, owned by the platform engineering team, and Score files, which developers use to request resources their workloads depend on, the Platform Orchestrator creates a Resource Graph which defines the order in which resources should be provisioned during a deployment.

In summary, the deployment pipeline is fully automated and aims to provide a streamlined experience for developers, removing the complexity of infrastructure management and enabling them to deliver value quickly and efficiently. The system ensures both speed and quality by incorporating continuous testing and observability.

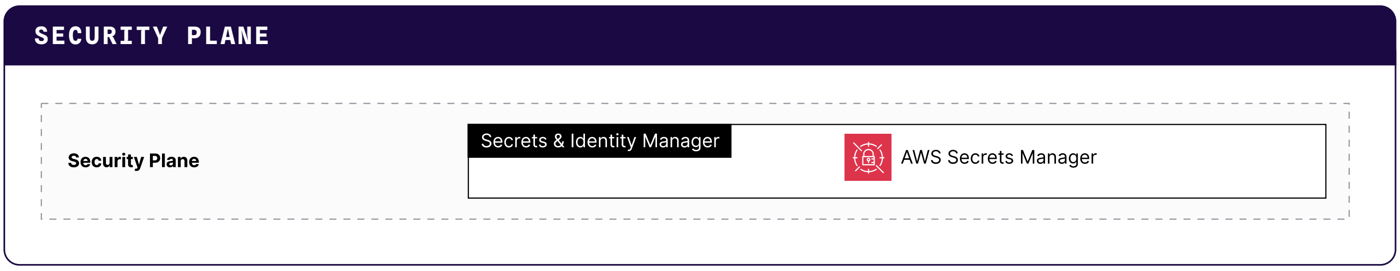
Monitoring and Logging Plane

Using tools like Datadog, the Monitoring and Logging Plane focuses on the stability of the system, tracking uptime, SLOs, and using RED metrics.



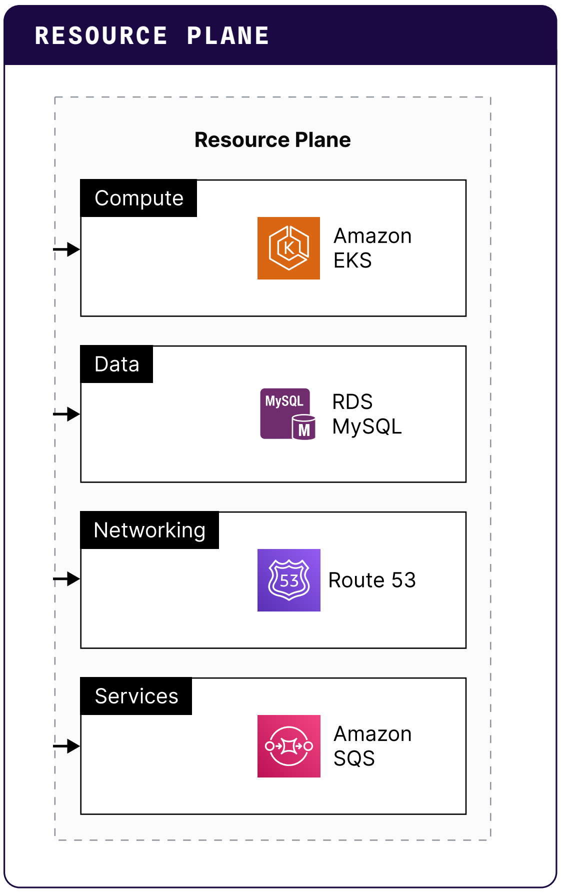
Security Plane

Convera's Security Plane is designed to ensure that all sensitive information is managed securely and consistently across the platform. The core component of this plane is AWS Secrets Manager, which stores critical configuration information like database passwords, API keys, and TLS certificates. Access to these secrets is tightly controlled, with EKS pods granted access only to the secrets they require, minimizing the potential impact of security breaches.



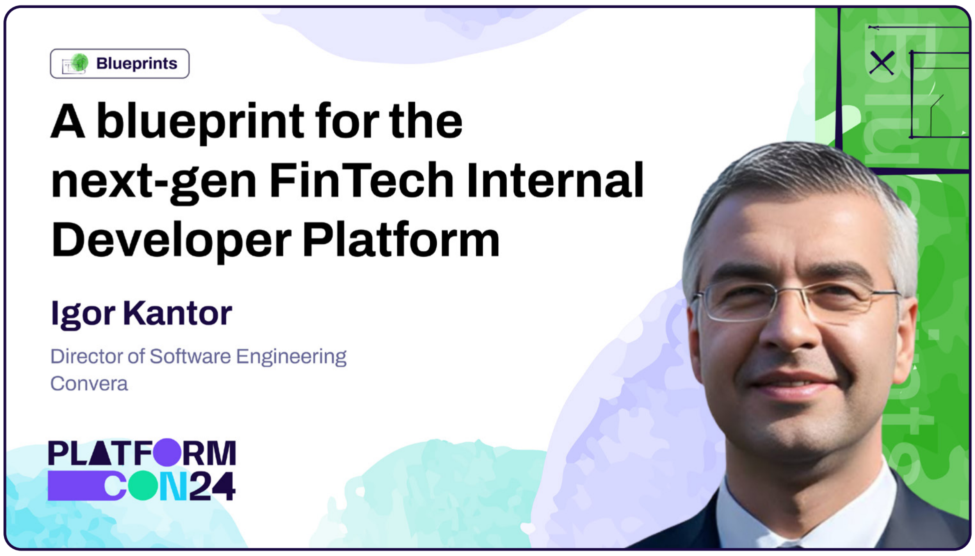
Resource Plane

This plane is where the actual infrastructure exists and includes clusters, databases, storage, or DNS services. The configuration of the Resources is managed by the Platform Orchestrator which dynamically creates app and infrastructure configurations with every deployment and creates, updates, or deletes dependent Resources as required.



Convera's reference architecture: A blueprint for Internal Developer Platforms

The reference architecture we presented in this whitepaper offers a blueprint for building an Internal Developer Platform (IDP) on AWS with GitLab, Humanitec, Terraform, and Datadog. It aims to tackle challenges like slow time to market, operational inefficiency, poor developer productivity, and lack of standardization.


Blueprints
A blueprint for the next-gen FinTech Internal Developer Platform
Igor Kantor
Director of Software Engineering
Convera
PLATFORM CON24

As Igor pointed out in his PlatformCon talk, Convera undertook a transformation to evolve into a leading-edge FinTech organization by embracing platform engineering. This shift was driven by the necessity to move away from outdated practices and promote innovation.

To achieve this, Convera concentrated on three key objectives: streamlining development, empowering developers, and ensuring operational efficiency. Platform engineering, specifically through building an internal developer platform (IDP), was implemented to realize these principles. Streamlining the software development lifecycle (SDLC) involved implementing continuous integration and continuous delivery (CI/CD), Infrastructure as Code (IaC), and continuous quality assurance, with feature branch testing to safeguard shared environments. Empowering developers was facilitated by removing complexity and enabling self-service using tools like the Score. To achieve operational efficiency, Convera concentrated on using cloud-native technologies, such as containerization, which enabled immutable deployments and the ability to scale with load, while also ensuring that cost was considered as a key fitness function. Ultimately, platform engineering at Convera aimed to establish a fully automated digital supply chain, emphasizing seamless workflows, empowered developers, and operational efficiency, thereby treating the platform as a product and developers as valued customers.

Convera's transformation highlights the power of platform engineering in streamlining development, empowering developers, and driving operational efficiency. By embracing cloud-native technologies, self-service tooling, and automation, they built an Internal Developer Platform that accelerates innovation while maintaining cost efficiency. If you're looking to achieve the same results as Convera and build a strong foundation in platform engineering principles, the Platform Engineering Fundamentals course is a great place to start. At the same time, for organizations looking to upskill entire teams and implement these best practices at scale, our Trainings provide tailored programs to help you build, optimize, and manage high-performing platforms effectively.

PlatCo GmbH

Wöhlertstraße 12-13, 10115 Berlin, Germany

Phone: +49 30 6293-8516

PlatCo Inc

228 East 45th Street, Suite 9E,

New York, NY 10017

PlatCo Ltd

3rd Floor, 1 Ashley Road

Altrincham, Cheshire WA14 2DT

United Kingdom

E-mail: info@platco-group.com

Website: https://platco-group.com

CEO: Kaspar von Grünberg

Registered at Amtsgericht Charlottenburg, Berlin: HRB 262650

VAT-ID according to §27a UStG: DE367439464

Responsible for the content of humanitec.com ref. § 55 II RStV: Kaspar von Grünberg