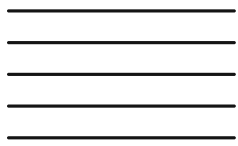




Guide

# Building Internal Developer Platforms: Best Practices, Tools, and Strategic Insights for Tech Leaders



# Executive Summary

This whitepaper provides a comprehensive overview of Internal Developer Platforms (IDPs). As engineering teams face growing complexity, IDPs are a solution to reduce friction between development and operations.

Whether you're evaluating platforms or building one in-house, this whitepaper offers insights and curated resources to help you navigate your IDP journey effectively.

## Takeaways:

- IDPs are self-service platforms built by platform engineers to streamline software development by automating deployments, configurations, and environment management.
- Benefits: Reduced time spent on setup and debugging, reduced cognitive load on developers, improved reliability and compliance and higher operational efficiency.
- Each IDP is unique, but they usually consist of a frontend, backend and other integrated tools.
- The core capabilities of an IDP should be: API, user interface, automation, constraints like policies, documentation, and discoverability.
- To build a truly cohesive IDP, a core focus has to be on integrating all the different tools and services that are needed.
- When implementing an IDP these challenges may arise: Trying to do everything at once (and archiving nothing), lack of resources and budget, not being able to maintain the software catalog, and lack of communication between people.
- When making the case for an IDP, you should support it with data and connect to the risks and outcomes your executives care most about.



# Table of Contents

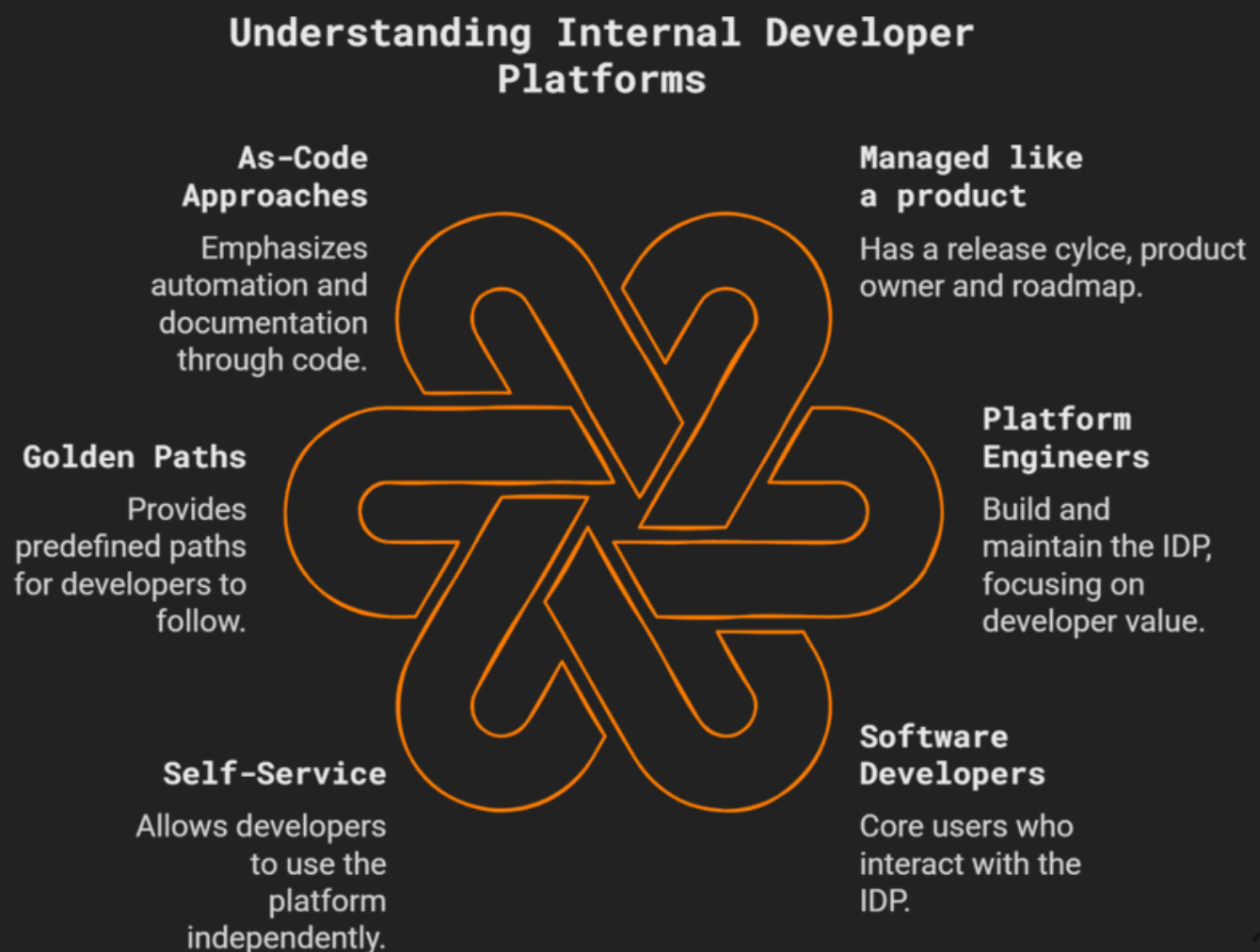
|  |    |
|--|----|
| What is an IDP                                 | 2  |
| Why IDPs                                       | 3  |
| How do IDPs work?                              | 4  |
| Making the Case for IDPs                       | 7  |
| Best Practices for Building an IDP             | 14 |
| Platform Engineering Tools for Building an IDP | 17 |
| 3 Challenges when building your IDP            | 25 |
| Resources                                      | 28 |

# What is an IDP?

Internal Developer Platforms (IDPs) are at the core of the platform engineering discipline.

They are self-service platforms built by platform engineers to streamline software development by automating deployments, configurations, and environment management.

IDPs aim to solve the perennial crux that haunts all software developers: Software is hugely complex and no one person can know everything that is required to build an entire software product.



IDPs provide features that are central to the daily work of software developers.



These are just examples of features that an IDP can have. IDPs are always unique to the organisation that is using it. They are often hand-built from scratch, or heavily customised.

Want to go deeper? Read our blog post: [What is an IDP?](#)

# Why IDPs?

As software delivery grows more complex, organizations struggle with fragmented DevOps practices, inconsistent workflows, and operational inefficiencies. Developers are expected to manage infrastructure, CI/CD pipelines, security policies, and monitoring, often leading to cognitive overload and reduced productivity. This is where Platform Engineering and an IDP comes in.

## Unveiling the Power of IDPs



# How do IDPs work?

Each IDP is unique, but there are some underlying characteristics that most of them share. Very basically, each IDP consists of approximately three “parts”:

- An IDP frontend
- An IDP backend
- Lots of other tools that are integrated with the IDP

## IDP Components



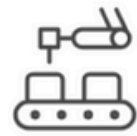
### Frontend

A user interface for developers to access the IDP.



### Backend

Manages integration and automation with other tools.

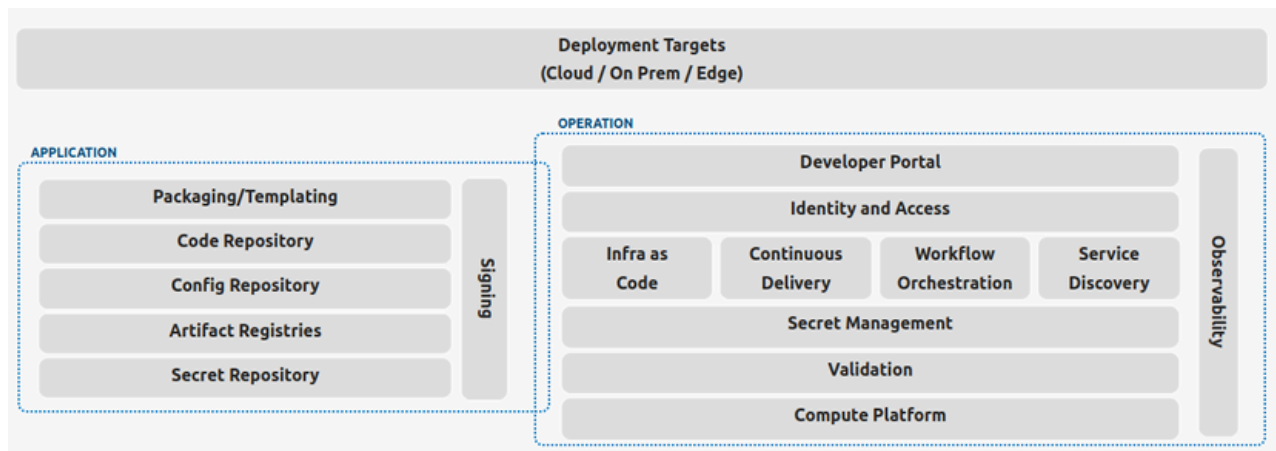


### Integrated Tools

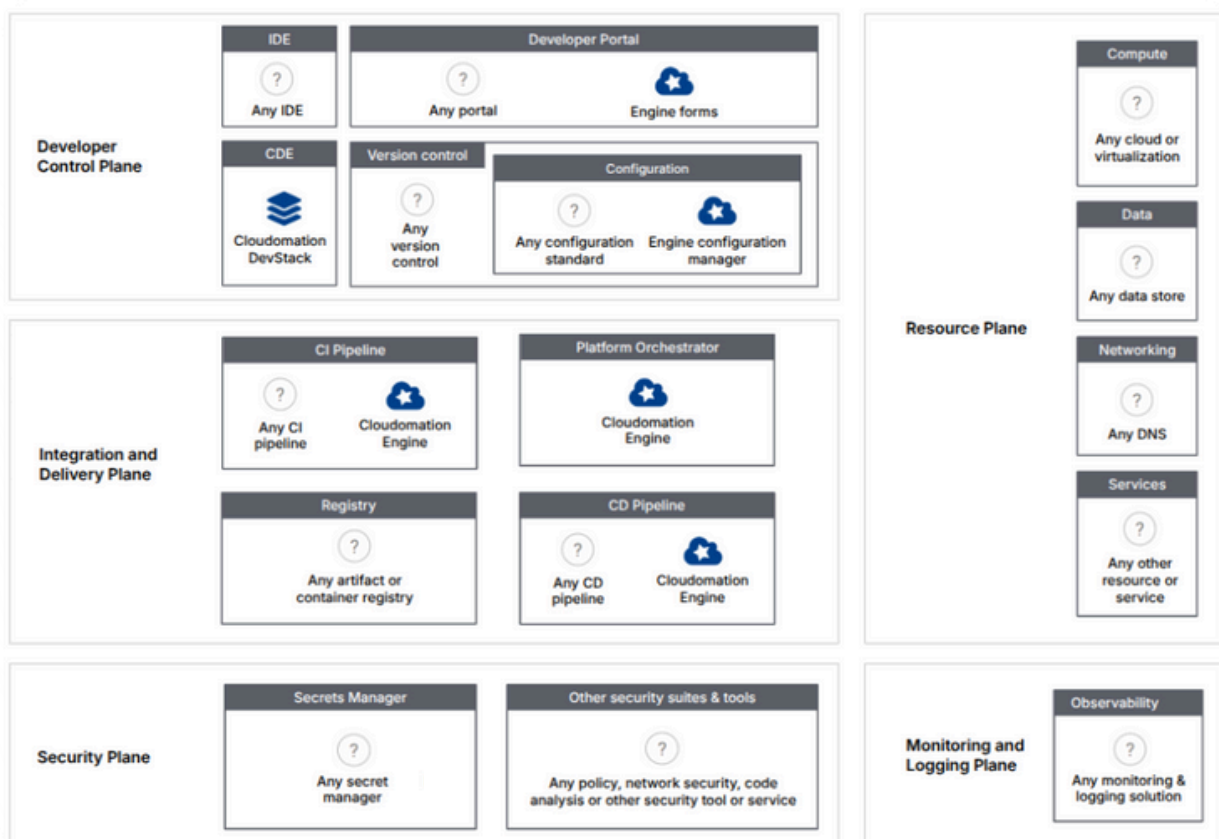
Various tools that work with the IDP for core processes.

# Architecture

The following architecture diagram from cnoe (Cloud Native Operational Excellence) gives a more detailed overview:



In this diagram, the “Developer Portal” would be the frontend of the Internal Developer Platform. The “Workflow Orchestration” bit would be the backend of the Internal Developer Platform. Below is a more detailed example architecture featuring our own products and showing which other types of tools and services could be part of an IDP:

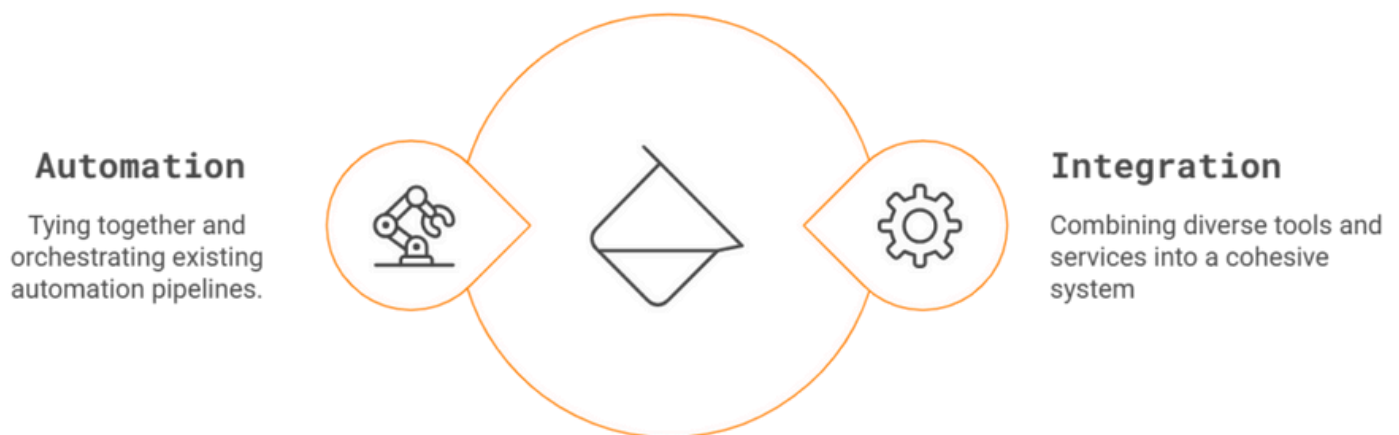




## Core functionality of an IDP

The main idea of an IDP is to tie together tools, services, configurations and other information and assets in one place. Software Engineers, but also other stakeholders e.g. in operations teams should be able to use the IDP as a single entry point to discover and interact with a company's applications and infrastructure.

As such, the IDP backend typically needs to be strong in two types of functionality: #1 Integration and #2 Automation.



Want to go deeper? Read our blog post: [How do IDPs work?](#)

# Making the Case for an IDP

*Based on insights from this informative video "How to Make the Business Case for an Internal Developer Platform"*

As organizations scale and modernize their software delivery, the complexity of managing infrastructure, developer workflows, and governance grows exponentially.

An Internal Developer Platform (IDP) offers a strategic response to this challenge, enabling faster delivery, stronger governance, and lower operational risk. However, to gain executive buy-in, it's crucial to frame the IDP in business terms.

Here's how to make the case across four key executive concerns: Scale, Cost, Risk, and Governance.



## 1. Scale: Empowering the Organization to Move Faster

### **Executive Concern:**

How do we scale our engineering teams and delivery without bottlenecks?

As your organization grows, so does the pressure on application teams to deliver more features, faster. But without the right support, engineers take on infrastructure tasks outside their core focus, slowing down velocity and increasing cognitive load.

### **IDP Justification:**

An IDP boosts productivity by abstracting away repetitive, low-value tasks. It empowers app teams to self-serve infrastructure, CI/CD, testing, and deployment in a standardized, secure way.

### **Business Outcomes:**

- Increase velocity of application teams and the organization as a whole.
- Deliver more business value, faster, because the development team can focus on coding.
- Enable scaling without linear increases in DevOps hiring.

### **How to do it:**

Asses how much time application teams spend struggling with infrastructure and deployment tools and how much time they spend working on their primary application. Use this data to make the case for an IDP and how it can help to free up time for the main task, which directly impacts business value. *Show this visually to highlight how much infrastructure has to be managed by application teams and contrast it with what it could look like.*

**PLATFORM CON23**

### Example pitch - Scale (Current state)

The diagram illustrates the current state of the platform architecture. It is divided into two main layers: the **Application team** (top) and the **Platform team** (bottom), with a **50 app teams** label indicating the scale. The Application team layer includes components like App development, Test frameworks, sonarqube (Code quality / scanning), Prometheus (Metrics collection + monitoring), Application runtimes, SENTRY (Error reporting), Infrastructure + deployment configuration, and Automated deployment pipelines. The Platform team layer includes GitLab CI pipeline, Package repository (GitLab + CodeArtifact, ECR), and GitLab Code repository. A central box labeled **Not in place** indicates missing capabilities: Distributed tracing, Log aggregation, Secrets management, Compliance automation / policy-as-code, and Automatic cloud network and access configuration. An upward arrow points from the Platform team to the Application team. A small video inset shows a person speaking.

**Kendra Skeene**  
Director of Product  
Ad Hoc LLC

**Andrew Gunsch**  
Director of Engineering  
Ad Hoc LLC

**PLATFORM IMPACT**

5:46 - Show the current state

**PLATFORM CON23**

### Example pitch - Scale (Future state)

The diagram illustrates the future state of the platform architecture. It is divided into two main layers: the **Application team** (top) and the **Platform team** (bottom), with a **50 app teams** label indicating the scale. The Application team layer includes components like Product management, App development, and Research + design. The Platform team layer includes GitLab CI pipeline, Test frameworks, Package repository (incl. trusted base images), Distributed tracing, Secrets management, GitLab Code repository, Container orchestration, Log aggregation, Prometheus (Metrics collection + monitoring), Compliance automation / policy-as-code, Deployment pipeline, IaC (Terraform configuration), Automatic cloud network and access configuration, and SENTRY Error reporting. An upward arrow points from the Platform team to the Application team. A small video inset shows a person speaking.

**Kendra Skeene**  
Director of Product  
Ad Hoc LLC

**Andrew Gunsch**  
Director of Engineering  
Ad Hoc LLC

**PLATFORM IMPACT**

6:36 - Show the future state

## 2. Cost: Optimize Through Scale and Reuse

### **Executive Concern:**

How do we control rising cloud and engineering costs?

Platform engineering reduces duplication of effort and consolidates infrastructure management. An IDP facilitates cost-effective scaling, enabling visibility and control across the software lifecycle.

### **IDP Justification:**

Centralization enables resource sharing, proactive cost monitoring, and automatic reclamation of unused assets.

### **Business Outcomes:**

- Improved cost visibility and predictability.
- Lower total cost of ownership (TCO) through resource optimization.
- Greater ROI on cloud infrastructure investments.

### **Highlight:**

Platform teams can track usage metrics and turn off unused environments.

### 3. Risk: Reduce Failure Points and Operational Hazards

“Risk mitigation” was one thing the speakers found particularly effective in pitching platforms to management.

#### **Executive Concern:**

What risks threaten our ability to deliver reliably and securely?

With fragmented infrastructure and inconsistent processes, risk multiplies. From compliance violations to production outages, decentralized practices amplify exposure.

#### **IDP Justification:**

An IDP minimizes operational and security risks by standardizing processes across the board—CI/CD pipelines, deployment, observability and alerting, and authentication.

#### **Business Outcomes:**

- Reduce the risk of production failures
- Centralize key services like deployment, identity, and logging.
- Enforce security and compliance policies from the platform layer.

#### **Example Risks Addressed:**

- Shipping delays due to flaky pipelines.
- Data leaks from misconfigured auth systems.
- Audit failures from inconsistent logging or compliance drift.

## 4. Governance: Enabling Consistency and Compliance at Scale

### **Executive Concern:**

How do we maintain control without slowing teams down?

As teams grow, ensuring compliance with organizational standards becomes increasingly difficult. Governance is often reactive rather than proactive.

### **IDP Justification:**

With an IDP, governance becomes baked in. From templates to APIs, platform teams can encode standards and best practices directly into the developer experience.

### **Business Outcomes:**

- Proactive governance and policy enforcement.
- Improved auditability and regulatory alignment.
- Foster a culture of engineering excellence.

## Supporting the Case with Data

A key differentiator in your business case is data, especially when comparing platform teams vs. application teams. Survey your engineering organization to gather insights on e.g. time spent on infrastructure vs. feature development.

## Final Thoughts: What Keeps Execs Up at Night?

When framing your case, connect directly to the risks and outcomes your executives care most about. For example:

- Speed to market: Can we deliver faster than competitors?
- Reliability: Will our systems hold under pressure?
- Security & Compliance: Are we vulnerable to breaches or audits?
- Scalability: Can we grow without losing control?

An Internal Developer Platform isn't just a technical tool, it's a strategic asset. With the right framing, data, and vision, you can demonstrate how a well-executed IDP supports not just engineering, but the entire business.



# Best Practices for Building an Internal Developer Platform



**Margot Mückstein** ✓ • 1.

Cloudomation powers software integration and automation across n...  
2 Wochen • 🌐

...

This is a post for people who know what platform engineering is and are looking for a structured way to go about building a maintainable and useful developer platform

I called it "best practices for building an internal developer platform", but it really is a list of the core capabilities that an IDP should have. It was inspired by a video by [Viktor Farcic](#) called "From Zero to Fully Operational Developer Platform in 5 Steps!" - I'll add the link to the video in a comment on this post.

Victor Farcics [video](#) was the first resource that helped us at Clouddomation how to go about building an IDP: Where to start and what's important.

In the video, [Victor Farcic](#) describes how to build a developer platform in 5 steps:

- API,
- State management
- One-shot actions / Workflows
- RBAC & Policies
- User interfaces (Optional)

We find these steps useful, not so much as steps, but because they describe the core capabilities a developer platform should have in order to be useful. They don't try to say what a platform should look like by listing product categories, but instead what a platform should be able to do. How you achieve those capabilities is totally up to you.

Here is our own take on the core capabilities of IDPs, which was inspired by the video. However, we formulate them slightly differently and added another one:

- API
- User Interfaces (not optional)
- Automation: One-shot Actions and State Management
- Constraints (policies, RBAC etc.)
- Documentation and Discoverability

How does this help me build a platform?

If you think about the problems you have had recently as a platform or DevOps engineer, you will probably be able to map them to one of the described capabilities.

Some examples:

- If you have trouble with downtimes of instances, or accidental high cost due to cloud resources being left lying around, you have insufficient state management capabilities.
- If you have issues with software engineers not having access to infrastructure they need, or with junior engineers messing with deployments they should not mess with, you are not managing constraints well.
- If your developers are unable to spin up a feature branch system on their own, or triggering a build and test run for a specific commit, then you need to take a look at your one-shot-action automation capabilities.
- If your software engineers are simply not using the APIs and services you provide, you should probably take a look at the user interfaces you provide or check if your services are discoverable / documented.
- If you have policies flying around everywhere and no real idea which constraints apply where and if it makes sense, or if everyone simply has full rights on everything and you know that this is not how it should be but simply don't know how to sustainably manage a principle of least privilege across your entire infrastructure and user base, then you should look for a tool that makes constraint modeling simpler for you.

When you start thinking about the problems you have - or, ideally, the services you would like to be able to provide to your software engineers - in these categories, it will help you identify what you need to do, and how to go about doing it in a way that will make your life substantially easier.

The purpose and value of these core capabilities is to help you understand why you keep failing with some topics, and how you can start fixing things in a way that lasts and is sustainably manageable.

Want to go deeper? Read our blog post: [Best practices for building an IDP](#)



# Platform Engineering Tools for Building an IDP

This article walks through how to structure an IDP, categorized by key components, with examples of tools you can use.

## Categorization of tools and components

We categorize these tools using a “reference architecture” popularized by [platformengineering.org](https://platformengineering.org). This framework breaks down the ecosystem into five core components, known as “planes”:

- **Developer Control Plane:** All components through which developers interact with the platform. Typically includes GUIs / portals, source control, cloud development environments (CDEs), as well as configuration standards that developers maintain themselves (typically in their source code repository), such as ansible playbooks, helm charts, devfile.yaml etc.
- **Integration & Delivery Plane:** This is where all the automation happens. CI/CD pipeline tools, infrastructure automation, as well as other automation tools (e.g. platform orchestrators) are typically shown as part of this plane.
- **Monitoring & Logging Plane:** As the name suggests, this is where all observability tools are located.
- **Security Plane:** Secrets management, policy tools and other security tools.
- **Resource Plane:** Compute and storage.

Not all internal developer platforms necessarily have all five planes, and it doesn't always make sense to divide an IDP into those five planes, e.g. when some of them are covered in the same tool or platform.

Next, we show you an example of how to design an IDP and the tools you could consider.

## Example: Building an Internal Developer Platform

Here's a breakdown of tools you could use to build an IDP. Important note: There are many tools available. Those referenced in this article are only examples.

### Developer Control Plane

#### #1 Developer Portal

Why it's important: Internal developer portals serve as the unified interface and allow developers, teams, and engineering managers to discover services, track ownership, enforce standards, and improve software. We wrote about Developer Portals in detail in this blog post [5 Internal Developer Portals \(...and what software engineers say about them\)](#). The platform engineering team ensures that the portal stays up to date, integrates seamlessly with existing tools, and evolves based on developer feedback.

Tools to consider:

- **Engine Forms:** Engine forms are lightweight, json-schema-based forms which can serve as simple user interfaces to e.g. trigger a deployment, provide information about pipeline status, etc. Engine forms are useful for fast prototyping and as simple user interfaces for individual use cases, however they are not intended as a full portal solution. For an IDP, a dedicated portal makes sense, to which Cloudomation Engine can expose data and services, which developers then consume via the portal.
- **Backstage:** Backstage is a popular open-source framework for building developer portals.
- **Port:** Port offers a no-code setup that makes it easy to get started quickly.
- **Cortex:** "Cortex is the enterprise Internal Developer Portal built to accelerate the path to engineering excellence."

## #2 Cloud Development Environments (CDEs)

Why it's important: CDEs are remote development environments that are either hosted in the cloud, or self-hosted. CDEs allow developers to work from consistent, standardized environments that eliminate "it works on my machine" issues. The platform engineering team ensures that CDEs are secure, fast, and integrated with the developer workflow.

Tools to consider::

- **Cludomation DevStack:** Cludomation DevStack Cloud Development Environments (CDEs) are fully equivalent to local development environments. Complex applications can be run directly in the CDE. The source code can be mirrored locally, which means that local IDEs can be used. Developers do not have to change existing working methods, but can work with CDEs as they would locally – only with more resources and without troubleshooting local deployments.
- **Gitpod:** With Gitpod, you can launch secure, context-rich environments built for developers and their agents at enterprise scale.
- **Coder:** Coder provides secure environments for developers and their agents.

We wrote an article about available CDE tools here: [7 Remote Development Tools at a Glance](#).

We've also put together a comprehensive [whitepaper covering all the major tools and vendors, complete with a detailed comparison table](#). As far as we know, it's the only resource offering such an in-depth comparison of CDEs.



## Integration & Delivery Plane

### #1 CI Pipeline

Why it's important: Automates code validation, testing, and faster feedback loops.

Tools to consider:

- **Cloudomation Engine:** Natively build CI pipelines from scratch, or migrate CI pipelines from other tools, integrate existing tools, orchestrate and extend existing pipelines.
- **GitHub Actions:** Native CI for GitHub, with customizable workflows.
- **CircleCI:** Highly scalable, with support for advanced parallelism.
- **Buildkite:** Developer-centric CI with scalable infrastructure.

### #2 CD Pipeline

Why it's important: CD pipelines automate the safe, repeatable deployment of software.

Tools to consider:

- **Cloudomation Engine:** End-to-end deployment automation. Automate complex deployment logic using Python. Gain full visibility with visualized deployment processes.
- **Argo CD:** GitOps-based delivery for Kubernetes.
- **Flux:** Kubernetes GitOps controller.
- **Octopus Deploy:** Suited for multi-cloud and on-prem environments.

### #3 Platform Orchestrator

Why it's important: Orchestration tools provide the logic to tie workflows together across tools and services.

Tools to consider:

- **Clouddomation Engine:** Pure Python Framework for Platform Engineering. Provide self service tools, automate complex tasks and get full visibility into your infrastructure with just one tool.
- **Humanitec:** A platform orchestrator providing dynamic environments.

### Monitoring & Logging Plane

#### #1 Observability

Why it's important: Observability tools provide visibility into the health, performance, and reliability of applications and infrastructure. The platform engineering team maintains dashboards, alerting rules, and ensures that teams get actionable insights.

Tools to consider:

- **Prometheus:** Monitoring and alerting toolkit, especially for Kubernetes.
- **Grafana:** Dashboarding tool often paired with Prometheus.
- **Datadog:** Cloud monitoring and analytics.

## Security Plane

### #1 Secret Manager

Why it's important: Secret managers securely store and distribute credentials, API keys, and other sensitive data.

Tools to consider:

- **HashiCorp Vault:** Industry-leading secrets management.
- **AWS Secrets Manager / Azure Key Vault / GCP Secret Manager**
- **Sealed Secrets (Bitnami):** Encrypts secrets for Kubernetes.

## Conclusion: Building Blocks, Not a Shopping List

If there's one thing to take away from this chapter, it's that platform engineering isn't about picking the flashiest tools off a shelf, it's about curating the right building blocks to create a seamless developer experience. Think less about "which tool should I pick?" and more about "how can I design a platform that feels invisible and powerful to my developers and provides business value?"

Because the real magic happens when these tools stop being individual puzzle pieces and start becoming part of a cohesive developer platform, where developers barely notice the underlying complexity because the platform works with them, not against them.

To build a truly cohesive IDP, a core focus has to be on integrating all the different tools and services that are needed. A painful lesson that many platform teams learn early on is that the feature set of individual tools is far less important than the ability to connect these tools with the IDP, and to provide it to software engineers with a good user experience - because otherwise, even the flashiest tool will just sit on the shelf and gather dust. As such, a central integration component (like a platform orchestrator) is key to building a successful internal developer platform.

# 3 Challenges when building your IDP

Guy Menahem is a Solutions Architect at Amazon (as of 02.05.2025), the Co-Founder of “Platformers Community” and a CNCF Ambassador.

In a recent video, he explains 3 real challenges when building your IDP, and we’ll walk through them below. However, there’s one challenge he doesn’t mention, and that’s what we’ll cover in our take.

## Challenge 1: Getting zero birds with one stone

Many platform engineering teams try to do everything at once and end up achieving nothing. Combining all tools into a single solution can cause you to miss the main user workflow, leading to platform abandonment.

Instead, sharpen your product skills to understand how users will benefit from the platform, even if it involves only a few tools initially.

Answer questions about user challenges, stakeholders, and their needs to define a successful platform for your company.



## Challenge 2: Estimation of build and operational costs

Delivering a valuable platform requires resources, including a dedicated platform engineering team, cloud infrastructure budget, and collaboration. Operational costs, such as upgrades and security patches, also need consideration.

Estimate resources by determining the team size and duration, focusing on a minimum viable product (MVP), and calculating cloud and operational costs.

## Challenge 3: Building and managing the software catalog

Managing a software catalog, which is a complex database of software, systems, and documentation can be challenging.

Getting all teams involved in updating and maintaining the catalog is crucial for its quality and adoption.

Make catalog management easier by automating information fetching, enforcing updates during CI/CD processes, and encouraging daily use of the platform.

## Our take

The challenges Guy describes are real.

One fundamental truth he doesn't mention, however, is the fact that most challenges when building an IDP are not technical.

Instead, the most common challenges come from a lack of communication between people. This is typical for many engineering initiatives, since engineers tend to have a very specific perspective on the tools they are building, and that perspective is often very different from the perspective of their users.

The same often happens in platform engineering teams: They may build valuable automation and services, but if those are not easy to use, miss core features that software engineers need, or simply don't address the biggest issues software engineers have, then software engineers will simply not use the IDP.

Guy does mention some aspect of this in the first challenge "*zero birds with one stone*":

Engineering an IDP that misses the point because it tries to do everything at once. Whether you try to do everything at once, or you are focusing on solving a single problem: the most important thing you need to do as a platform engineer is ask your software engineers - regularly! - to test what you're building, to tell you if it is useful for them, and if the answer is no, then to change whatever it is you're building so that your software engineers will want to use it.

After all, that's the entire point of an IDP: It has to be useful for software engineers.

# Resources

## Websites

<https://cloudomation.com>  
<https://platformengineering.org/>  
<https://internaldeveloperplatform.com>  
[infoq.com/platformengineering](https://infoq.com/platformengineering)  
<https://pemonthly.com/>

## YouTube

Cloudomation - [youtube.com/channel/UCUG5PJEmYyZZws4Bh8JWFsg/](https://youtube.com/channel/UCUG5PJEmYyZZws4Bh8JWFsg/)  
DevOps Toolkit - <https://www.youtube.com/@DevOpsToolkit/videos>

## Communities

Reddit - [https://www.reddit.com/r/platform\\_engineering/](https://www.reddit.com/r/platform_engineering/)  
Platformers - <https://platformers.community>


## Newsletter

<https://platformweekly.com>

## Curated list of tools and resources

[5 Internal Developer Portal Tools](#)  
[Platform Orchestration Tools](#)  
[Platform Engineering Tools](#)





# Meet Your New Platform Engineering Tool

Provide self service tools, automate complex tasks and get full visibility into your infrastructure with just one tool – Cludomation Engine.

Book a free call



Cludomation is a brand of  
Starflows OG  
Darnautgasse 6/6  
1120 Vienna, Austria  
<https://cloudomation.com>  
[info@cloudomation.com](mailto:info@cloudomation.com)