Platform Engineering Maturity Model

Presented by the Cloud Native Computing Foundation Working Group Platforms

Version 1.0.0
Released October 31st, 2023

Introduction

CNCF's initial Platforms Definition white paper describes what internal platforms for cloud computing are and the values they promise to deliver to enterprises. But to achieve those values an organization must reflect and deliberately pursue outcomes and practices that are impactful for them, keeping in mind that every organization relies on an internal platform crafted for its own organization - even if that platform is just documentation on how to use third party services. This maturity model provides a framework for that reflection and for identifying opportunities for improvement in any organization.

Table of Contents

What is platform engineering?

Inspired by the cross-functional cooperation promised by DevOps, platforms and platform engineering have emerged in enterprises as an explicit form of that cooperation. Platforms curate and present common capabilities, frameworks and experiences. In the context of this working group and related publications, the focus is on platforms that facilitate and accelerate the work of internal users such as product and application teams.

Platform engineering is the practice of planning and providing such computing platforms to developers and users and encompasses all parts of platforms and their capabilities — their people, processes, policies and technologies; as well as the desired business outcomes that drive them.

Please read the CNCF Platforms Definition white paper first for complete context.

How to use this model

As platform engineering has risen in prominence over the last few years, some patterns have become apparent. By organizing those patterns and observations into a progressive maturity model, we aim to orient platform teams to the challenges they may face and opportunities to aim for. Each aspect is described by a continuum of characteristics of different teams and organizations at each level within the aspect. We expect readers to find themselves in the model and identify opportunities in adjacent levels.

Of note, each additional level of maturity is accompanied by greater requirements for funding and people's time. Therefore, reaching the highest level should not be a goal in itself. Each level describes qualities that should appear at that stage. Readers must consider if their organization and their current context would benefit from these qualities given the required investment.

Keep in mind that each aspect is meant to be evaluated and evolved independently. However, as in any socio-technical system these aspects are complex and interrelated. Thus you may find that to improve in one aspect you must reach a minimum level in another aspect too.

It's also important to recognize that implementations of platforms vary from organization to organization. Make sure to evaluate the current state of *your* group's overall cloud native transformation. A phenomenal resource to leverage for this evaluation is the Cloud Native Maturity Model.

Finally, this model encourages organizations to mature their platform engineering discipline and their resulting platforms through intentional planning. Such planning and discipline themselves are a requirement for mature platform development and ongoing evolution.

In general, keep in mind that mapping your organization into a model captures current state *to enable* progressive iteration and improvement. Martin Fowler says it well: "The true outcome of a maturity model assessment isn't what level you are at but the list of things you need to work on to improve. Your current level is merely a piece of intermediate work in order to determine that list of skills to acquire next." In that vein, seek to find yourself in the model then identify opportunities in adjacent levels.

Context behind this work

It's valuable to understand the context a document has been written in. The following sections lay out some context behind the model as well as some expectations for you, the reader.

Intended audiences

Each reader brings a unique context and will take unique learnings from this model. Following are some personas we have in mind, along with their possible motivations for engaging with this model:

- **CTOs, VPs, and directors of technology**: Leaders looking to map a path to digital transformation and greater developer productivity
- **Engineering managers**: Groups and individuals seeking to empower engineers to provide value with less overhead and higher efficiency
- **Enterprise architects**: Individuals navigating the modern technology landscape who seek a value- and solution-oriented perspective on technology problems
- **Platform engineers and platform product managers**: Teams and people seeking to build the best possible experience for platform builders and platform users
- **Product vendors and project maintainers**: Organizations and engineers wishing to design tools and deliver messages to enable users to succeed with platforms and capabilities
- **Application and product developers**: Platform users seeking to understand in more detail what they might expect of an internal platform

Understanding the levels

This model is not meant to classify an organization or platform team as wholly "Level 1" or "Level 4." Each aspect should be considered independently of the others; the characteristics of each level represent a continuum within that aspect but are not necessarily coupled to other aspects at the same level. Even more so, many organizations will see characteristics of more than one level being applicable across their teams and work. This is because no level is inherently good or bad, only contextual to the team's goals.

The labels for each level are intended to reflect the impact of platform engineering at your organization. As you recognize your organization at a given level you will gain insight into opportunities which follow at the next ones. Lower-numbered levels comprise more tactical solutions while higher-numbered ones are more strategic.

This yields a potential process for platform development and maturity similar to other digital product development: first recognize a problem and need for a new solution, next develop minimally-viable products as hypothesized solutions, third iterate to better solve the problem and ensure fit for your customers and finally scale and optimize the product to solve the problem for many teams and users.

Similar to the CNCF Cloud Native Maturity Model, this model highlights that successful

business outcomes can only be achieved through balancing people, process, and policy alongside technology. Notably, this model introduces aspects which are often not fully in the remit of a single internal team, but rather require cooperation across the engineering department and quite often the wider organization.

But it doesn't seem to fit

That's perfectly fine! All organizations and groups have dynamics and parameters that are specific to them.

Keep in mind that the goal of this paper isn't to prescribe a rigid formula, but rather a framework that you can apply to your circumstances. Every single word may not be relevant to you, but we hope the content will inspire you to introspect on your own platform journey, taking what makes sense and leaving the rest.

The objective of this model is to provide a tool to help guide platform engineering practitioners, stakeholders, and other interested parties on their journeys. Platform design and implementation is not an exact science, but rather depends on the needs of an individual project, an organization and a particular time and place.

Model table

Aspect		Provisional	Operational	Scalable	Optimizing
Investment	*How are staff and funds allocated to platform capabilities?*	Voluntary or temporary	Dedicated team	As product	Enabled ecosystem
Adoption	*Why and how do users discover and use internal platforms and platform capabilities?*	Erratic	Extrinsic push	Intrinsic pull	Participatory
Interfaces	*How do users interact with and consume platform capabilities?*	Custom processes	Standard tooling	Self-service solutions	Integrated services
Operations	*How are platforms and their capabilities planned, prioritized, developed and maintained?*	By request	Centrally tracked	Centrally enabled	Managed services
Measurement	*What is the process for gathering and incorporating feedback and learning?*	Ad hoc	Consistent collection	Insights	Quantitative and qualitative

Model Detail

Investment

How are staff and funds allocated to platform capabilities?

Investment in platforms and platform engineering is the process of allocating budget and people to build and maintain common capabilities. It is common for initiatives to be described as organically built from the bottom up, or driven through top down initiatives. In either case, it is the ability to invest sustained effort that drives high-impact work. This aspect captures how the scale and breadth of investment can impact platform success.

Level 1, Provisional — Voluntary or temporary

Individual capabilities may exist to provide common foundations for common or critical functionality. These capabilities are built and maintained out of necessity rather than planned and intentionally funded.

These capabilities are built and maintained by people assigned temporarily or voluntarily; no central funding or staffing are intentionally allocated to them. They depend on the current tactical requirements of their users.

Characteristics

- “Hit” or “tiger” teams are built to tackle urgent requirements. These teams are short lived and not assigned nor granted the time to provide long term planning and support.
- Migrations, improvements, or enhancements are often considered “nice to have” work items and rely on “research” or “hack day” efforts.
- Process improvements or automation may be introduced while tackling a new requirement such as an urgent security patch, however there is not support to build the solutions in a reusable or sustainable way.
- Employees complain of burn out and frustration with the amount of work they are doing outside their core role.

Example Scenarios

- There is a specific employee who is viewed as the test environment expert. While this employee means well, their attempt to enable better test environments despite limited investment has led to increased risk since there is no maintenance of their solution and no shared understanding of how to triage a broken test environment.
- Engineers are encouraged to invest in capability improvements when there is no pressure from management for revenue generating features. This translates to the last few days of some sprints where they prioritize automating and improving parts of their CI/CD pipeline. It is not uncommon for these improvements to come in bursts as there can be months of overly full sprints not allowing for time on these side endeavors.

Level 2, Operationalized — Dedicated team

Budget and people are allocated for persistent people and resource support. The assigned people are tasked with providing a set of commonly-required capabilities to speed up software delivery. Often these teams focus on meeting reactive technical requirements. They may be called DevOps, Engineering Enablement, Developer Experience (DevEx or DevX), Shared Tools, a Centre-Of-Excellence, or even Platform. They're funded centrally and treated as cost centers; their impact on direct value streams and application teams is not measured. It can be hard to map the impact of platform teams at this level on the organization and its value streams, which can make it hard to sustain and continue funding such teams.

Characteristics

- The team is made up of nearly all technical generalists.
- Team budget may include the infrastructure costs associated with their work leading to often being a key point in budget conversations.
- Backlog items range a number of technologies, leading to frequent and large context switches.
- This team is often the first to fill a gap that is not yet being addressed, even if not in the declared scope for the team. This team takes ownership of resources that don't have an owner.
- Assigned people rarely have the time or experience with customer research to validate their designs or implementations.

Example Scenarios

- Application developers raise an issue with the long build time for their applications. A centralized team is tasked with reducing the build time by 50%. They solve this by doubling the size and quantity of the CI runners given they are not close enough to the software to individually improve the application builds. This creates a budget concern for their centralized team as the productivity gain is not directly measurable against this increased infrastructure cost.

Level 3, Scalable — As product

Investment in internal platforms and their capabilities is similar to investment in an enterprise's outbound products and value streams: based on the value they are expected to provide to their customers. Product management and user experience are explicitly considered and invested in. A chargeback system may be used to reflect platforms' impact on their customers' own direct value streams and products. The enterprise allocates funds and staff to the appropriate initiatives by using data-driven performance indicators and feedback loops. Platform teams can ultimately optimize the business itself and contribute to increased profitability.

Characteristics

- Platform teams staff roles not traditionally found in internal serving or technical teams, for example, product management and user experience.
- The team publicizes a roadmap internally to the organization, which indicates the value delivered and high level feature targets.
- Features are tested for both implementation quality and user experience during design, delivery, and post deployment.
- Feature removal is a key part of the conversation, the goal is to have a well supported, well used suite of capabilities instead of a sprawling estate that may not be maintained.

Example Scenarios

- Data derived from platform usage metrics inform decisions to allocate funds and staff to the most impactful initiatives.

Level 4, Optimizing — Enabled ecosystem

Platform teams find ways to increase organization-wide efficiency and effectiveness beyond basic capabilities. Core platform maintainers intentionally strive to optimize time-to-market for new products, reduce costs across the enterprise, enable efficient governance and compliance for new services, scale workloads quickly and easily, and other cross-cutting requirements. These core maintainers are focused on enabling capability specialists to seamlessly integrate their requirements and offerings into existing and new parts of platforms. Further, the organization focuses people and resources from specialist domains like security, performance, quality on engaging with provided platform frameworks to introduce advanced features that can enable product teams to accelerate their adherence to company goals without depending on a centralized team backlog.

Characteristics

- It becomes a priority to enable specialists to extend platform capabilities and introduce new ones.
- The organization can centralize specialists allowing their knowledge and support to be spread through platform capabilities.

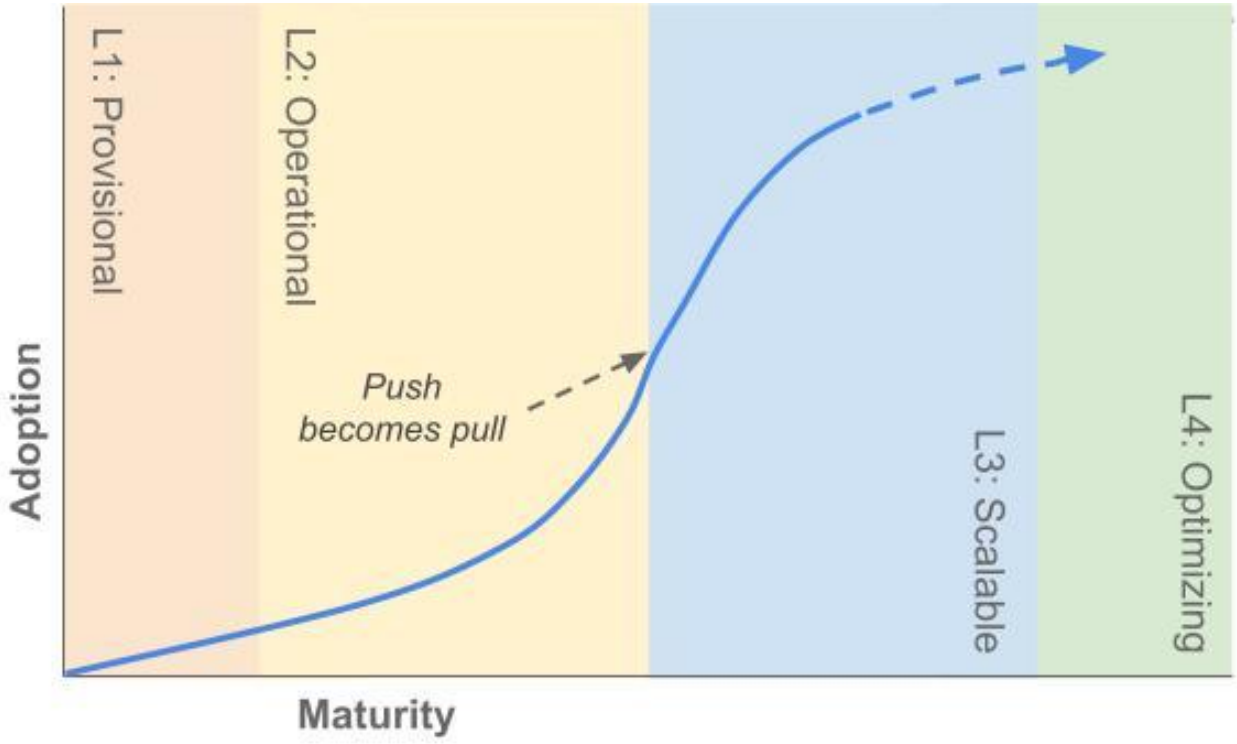
Example Scenarios

- Marketing works with platform builders to introduce consistent user tracking in order to attribute marketing efforts to product outcomes.
- Automation initiative reduces human time to provision databases by 30 minutes per instance, saving $10m/year.

Adoption

Why and how do users discover and use internal platforms and platform capabilities?

Adoption describes not only how and how much an organization uses platform capabilities, but also what motivates them to do so. In the early stages, many target users may not realize they are using a platform at all, rather they see their tools as an ad hoc collection of capabilities from various internal sources. This may mature into a group of capabilities that is consistently managed and presented to users — that is, one or more platforms. As the capabilities become more refined and discoverable, it is common that the drive for platform use moves away from more external motivations like mandates or incentives. This leads to users self-selecting into platform capabilities and ideally even investing their own efforts into the wider platform ecosystem.



A diagram to indicate a common growth pattern for platform adoption. This showcases the often slow start driven mainly by platform builders. Once platforms provide enough value to users, growth becomes more pulled by the users causing a steeper adoption curve.

Level 1, Provisional — Erratic

Adoption of shared platforms and capabilities is sporadic and inconsistent. No organization-wide strategy or guidance exists for choosing and integrating required backing services and technologies. Individual teams might leverage platform practices to improve their own processes, but there is no coordinated effort or standardization across the organization. This level of adoption is characterized by the absence of a coherent approach and the idea that external tools are more effective than those provided internally.

Characteristics

- One-off tools, services, and capabilities are managed by and consumed by various teams and departments in the organization.
- Provider-managed (aka “cloud”) services are adopted and used inconsistently and without standard practices and policies, as internal configurations are hard to find or use.
- App and service teams discover tools and capabilities haphazardly, via rumors and chance conversations rather than through a more centralized process.
- Coordination and reuse of components and capabilities is driven only by end users (application teams), if at all.
- Product teams each maintain their own set of scripts or tools to deploy their applications.

Example Scenarios

- A banking service requires a database. A developer finds out from a friend on another team that they can request an AWS account and set up an RDS database. From another team they find a Terraform script to provision that database. For monitoring they use CloudWatch on an ad hoc basis; they copy secrets from the AWS console to an instance of Hashicorp Vault manually before running the Terraform script.

Level 2, Operationalized — Extrinsic push

The organization recognizes the value of shared platforms and capabilities and strives to encourage and nurture them. Internal directives incentivize or even require use of shared platform services for some use cases. Some product teams use platform capabilities more than others; capabilities cover typical use cases in the organization but not unusual ones; and it is difficult to add those outliers to the common platform.

User discovery of capabilities and how to use them is inconsistent; it is possible a user on a product team won’t discover a supported capability unless directed there by a platform team.

Characteristics

- Some degree of external impetus leads to use of platform capabilities, for example:
    - Incentives such as personal reviews
    - Mandates such as requiring use for production releases or receiving funding
- The utilization of platform capabilities is fragmented — users may take advantage of one capability but might not be aware of, or interested in adopting, others that are available.
- Users have low motivation to learn how to use platform capabilities and rely heavily on collaboration with the providers through forums like office hours or help desk.
- Platform users are encouraged to join informal communities of practice to share problems and solutions but attendance may be limited.

Example Scenarios

- An engineering organization decides on a standard deployment tool and instructs all teams to use it. New processes (communication of release notes, etc) are built around that standard. Teams are instructed to stop using other sorts of deployment scripts and use the common tool instead. This is difficult for some teams whose needs are not met by the new process but do not understand or are not allowed to extend it.

Level 3, Scalable — Intrinsic pull

Users on product and service teams choose to use platforms and their capabilities because of the clear value they provide in reducing cognitive load on product teams while providing higher quality supporting services. Documentation and ergonomic interfaces enable product team users to quickly provision and use platform capabilities. Users choose internal platform implementations over alternatives such as developing the capability themselves or hiring a provider.

Characteristics

- Platform adoption is self-sustaining –The primary driver for core adoption is not an external impetus or incentive which mandates users use platform offerings – rather it is the values of these platform offerings themselves which draws users to them.
- After using and appreciating one or some platform capabilities, users seek out others and find the experience is similar across capabilities. There is an expectation that an individual capability is not isolated, rather it is one feature among a larger platform feature set.
- Platform teams encourage the natural adoption of platforms by gathering user feedback, sharing roadmaps and maintaining open forums for conversation with users.
- Application and product teams value platform capabilities enough to pay for them, e.g., via a chargeback system.
- Users can share feedback and learn about upcoming features through open forums and shared roadmaps.
- Self-serve portals, golden-path templates, and other documents enable rapid use.

Example Scenarios

- An application team previously had success requesting a new database. Their process was easy to understand and required almost no waiting time. In addition, key capabilities like backups and monitoring that allowed the team to progress their use all the way to production without issue were included. This experience meant that when the team later needed a queue, their first instinct was to check for an internal platform option. While they originally intended to use a specific queue technology, in the end, they chose to use the one offered internally since they knew how well integrated the solution would be for their organization.

Level 4, Optimizing — Participatory

Users from product teams further invest in platform capabilities by joining the ecosystem and contributing back to it. Some contributions improve and fix existing capabilities; others introduce new capabilities and features to address new use cases. Processes and services are defined and enable users to identify requirements and coordinate contributions amongst several product and platform teams. New capabilities are published via consistent interfaces and portals and with complete documentation and standard versioning.

Characteristics

- Users in app/service teams are empowered to contribute fixes, features, and feedback for platform capabilities.
- External projects and standards are strategically leveraged to reduce maintenance costs, accelerate new feature delivery, and use organization headcount most effectively.
- New capabilities and enhancements are coordinated asynchronously through issue boards and pull requests. Documents and checklists enable self-driven development by contributors.
- Developer advocates and internal ambassadors build and support an internal user community that extends platform ownership to app and service team contributors, too.
- Use of platform capabilities is viewed as the best way of working at the organization by both leadership and individual contributors.
- Platform engineers participate in product team planning to learn of requirements and suggest relevant existing capabilities.

Example Scenarios

- A team wants an alternative backup plan. After proposing this as a general offering, it is deemed low priority due to minimal reuse. The proposing team chooses to integrate their solution into the platform framework and make it available to the organization. It is originally an alpha offering but once it meets all of the operational requirements can be promoted to a core platform capability.

Interface

How do users interact with and consume platform capabilities?

The interfaces provided by platforms affect how users interact with these platform offerings to provision, manage, and observe capabilities. Interfaces can include ticketing systems, project templates, and graphical portals as well as automatable APIs and command-line (CLI) tools.

Key characteristics of an interface include how discoverable and user-friendly it is during key user journeys like initial request, maintenance, or incident triage. Higher levels of maturity here reflect more integrated, consistent, automated, and supported interfaces.

Level 1, Provisional — Custom processes

A collection of varying processes exists to provision different capabilities and services, but the consistency of the interface is not considered. Custom tailor-made processes address the immediate needs of individuals or teams and are reliant on manual intervention even if the provider uses some automated implementation scripts.

Knowledge of how to request these solutions is shared from person to person. The process for requesting a service lacks standardization and consistency. Provisioning and using a platform service likely requires deep support from the capability provider.

Lack of central requirements and standards makes this level appropriate when the company has not yet identified and documented expectations. It can be particularly effective for teams at early stage companies or platform efforts. In these environments teams are provided the freedom to evolve processes and capabilities to their needs, allowing them to deliver more quickly and pay the price of standardization only when necessary later on.

Characteristics

- User interaction is not a key topic of discussion and rarely (if ever) are interactions tested during design and delivery of new capabilities.
- Capabilities are mainly provided through manual requests, though providers may choose to automate some or all of the activities necessary to provision a user request.
- Requests that are on the face “simple” become complex due to finding out the right process to follow
- Sometimes a process appears to be sanctioned, but users run into issues when a different department or team gets involved

Example Scenarios

- An application team wants to performance test their new change. To do this, they want an isolated environment that contains enough test data to get an accurate performance read. The last time they had this request a former teammate was able to get access to an environment, but they have since moved on and no one knows how to recreate it. In the end, they are connected to an engineer on the infrastructure team who is able to provision them an environment in a few days.
- A team in the exploratory phases of product development uses a bespoke process to provision a new cloud service without needing to validate their solution warrants further investment.

Level 2, Operationalized — Standard tooling

Consistent, standard interfaces for provisioning and observing platforms and capabilities exist and meet broad needs. Users are able to identify what capabilities are available and are enabled to request capabilities that they require.

"Paved roads" or "golden paths", in the form of documentation and templates, are provided. These resources define how to provision and manage typical capabilities using compliant and tested patterns. While some users are able to use these solutions on their own, the solutions often still require deep domain expertise and therefore support from maintainers is still vital.

Characteristics

- Technical solutions are built-in tools specific to their problem domain, not always tools familiar to the users.
- There is investment in a common path; however, deviating from that path quickly uncovers few customization options as the focus was on building a single option.
- Given standardization, informal internal groups are able to form and gather to share good practices and overcome shared problems.
- There may be drift on capability implementation as teams take templates, customize them, and then cannot merge in changes from the centralized team.

Example Scenarios

- A centralized team curates a library of Terraform modules, Kubernetes controllers, and CRDs for provisioning different types of infrastructure.
- A shared location includes comprehensive documents about solutions across the organization.

Level 3, Scalable — Self-service solutions

Solutions are offered in a way that provides autonomy to users and requires little support from maintainers. The organization encourages and enables solutions to provide consistent interfaces that enable discoverability and portability of user experience from one capability to another. While self-service, the solutions do require team awareness and implementation. In order to improve this experience there may be a guided and simplified internal language which enables users to adopt and integrate platform capabilities more quickly. This generates a user-centric, self-serviceable, and consistent collection of capabilities.

Characteristics

- Solutions are provided as "one-click" implementations, enabling teams to benefit from a capability without needing to understand how they are provisioned.
- While the solutions are easy to create, there may not be as much usability built into the day 2 and beyond management of the solution.
- There continues to be a narrow path of available solutions, leaving users with unique requirements unsure how to proceed.

Example Scenarios

- An API is provided which abstracts the creation and maintenance of databases and provides users with any information they require to leverage that platform capability such as a connection string, location for secret data, and dashboard with observability data.

Level 4, Optimizing — Managed services

Platform capabilities are transparently integrated into the tools and processes that teams already use to do their work. Some capabilities are provisioned automatically, such as observability or identity management for a deployed service. When users hit the edges of the provided services, there is an opportunity to move past automated solutions and customize for their needs without leaving the internal offerings because platform capabilities are considered building blocks. These building blocks are used to build transparent and automatic compositions to meet the higher-level use cases while enabling deeper customization where necessary.

Characteristics

- It is clear what capabilities are differentiating for the organization and which are not, allowing the internal teams to invest in custom solutions only where they can not leverage industry standards.
- While capabilities are surfaced in a consistent way, there is no one way to use a capability. Some are best suited as CLI tools for use in scripts whereas others benefit from integration into where the user is writing code in their editors and IDEs.
- The value of individual capabilities is extended with a focus on the flow of both software development and release, leading to a focus on how to combine capabilities into higher level offerings.
- While capabilities are often provided in packages, super users are enabled to decompose these higher level offerings in order to optimize when and where they need to.

Example Scenarios

- Observability agents are injected into every workload and an OIDC proxy is placed in front of all applications.
- By default every new project receives a space in a task runner (pipelines) and a runtime environment (K8s namespace), however a project can opt into other options such as serverless runtime.
- From a catalog in a Service Now portal a user selects “Provision a Database.” Automation provisions an RDS database and sends a URL and location to get credentials to the user.

Operation

How are platforms and their capabilities planned, prioritized, developed and maintained?

Operation of platforms means running and supporting its capabilities and their features over their whole lifetime, including acceptance of new requests, initial releases, upgrades and extensions, ongoing maintenance and operations, user support, and even deprecation and termination. Organizations and their platform teams choose platforms and capabilities to create and maintain and can prioritize the most valuable and impactful initiatives.

Notably, most of the work to provide a capability is expended after its initial release — in providing seamless upgrades, new and improved features, operational support, and end-user enablement and education. Therefore an impactful, valuable platform will plan in advance and manage their platform for long-term sustainable operations and reliability.

Level 1, Provisional — By request

Platforms and capabilities are developed, published, and updated reactively, based on ad hoc product team requests and requirements. Product teams themselves may even need to plan and build the capabilities they require.

Teams who build a new capability, whether dedicated centralized teams or application teams meeting their own needs, take only informal responsibility for supporting others using it. They are not expected to actively maintain it and few processes exist to vet the quality of the offering. In this level, implementations are often ignored until a security vulnerability is discovered, a bug prevents use, or a new requirement arrives, at which point another reactive plan may be quickly implemented.

Characteristics

- Capabilities are created to meet the pressing needs of individual application teams.
- Focus is on initial delivery of core capabilities; plans are not made for ongoing maintenance and sustainability.
- Capability implementations are generally out of date and awaiting updates.
- Sudden spikes of work are introduced for late-breaking high-impact changes to capabilities, such as discovery of a vulnerability.
- Changes can result in both planned and unplanned downtime.
- Each upgrade is done in a bespoke way, requiring time and research to devise a process on each upgrade.

Example Scenarios

- Log4Shell security vulnerability is announced and the organization spins up a specialty team to investigate where the organization may be vulnerable and instigate patches. Once the team identifies the impact, they must work hand in hand with a number of different teams since each one manages their servers and upgrade processes differently. Even when this work is deemed complete, the confidence level is fairly low that there won't be more instances uncovered.

Level 2, Operationalized — Centrally tracked

Platforms and capabilities are centrally documented and discoverable, and processes for planning and managing the lifecycle of capabilities is at least lightly defined. Responsibility and ownership is documented for each service and function. Lifecycle management processes vary for different capabilities depending on their owners and their priorities. A centralized team maintains, or is able to on demand generate, an inventory of backlog capabilities to provide the state of maintenance for current capabilities. This allows the organization to track progress towards capability offering and compliance with upgrade requirements.

Characteristics

- Application teams create new capabilities as needed to meet pressing needs.
- A central team provides a register of available shared services across the organization.
- Loose standards, such as requiring an automatable API and usage docs, are applied to capabilities.
- Infrastructure as Code is used to allow easier traceability of deployed services.
- Audits for compliance regulations such as PCI DSS or HIPPA are enabled through the service inventories.
- Migration and upgrade work is tracked against a burndown chart enabling the organization to track rate of compliance and time until completion.
- Tracking does not indicate level of support; often upgrades at this stage are still manual and bespoke.

Example Scenarios

- PostgreSQL 11 is going EOL by the end of the year. The organization is aware of which databases require upgrade and are scheduling the work on each team's backlog to complete.

Level 3, Scalable — Centrally enabled

Platforms and capabilities are not only centrally registered but also centrally orchestrated. Platform teams take responsibility for understanding the broad needs of the organization and prioritize work across platform and infrastructure teams accordingly. Those responsible for a capability are expected to not only maintain it technically, but also provide standard user experiences for integrating the capability with other related services around the organization, ensure secure and reliable use, and even provide observability.

Standard processes for creating and evolving new capabilities exist, enabling anyone in the organization to contribute a solution that meets expectations. Continuous delivery processes for platform capabilities and features enable regular rollout and rollback. Large changes are planned and coordinated as they would be for customer-facing product changes.

Characteristics

- Application teams request services from platform teams first before creating them.
- New services must adhere to standard practices such as standard interfaces, documentation, and governance.
- Upgrade processes are documented and consistent across versions and services.
- Where the capability provider does not manage an upgrade, they provide tooling and support to the users for minimal impact.

Example Scenarios

- The organization is going to upgrade to RHEL 9. In doing so, each application team needs to validate that their software continues to work. In order to enable this testing phase the centralized compute team is setting up test environments for each team with the correct software and OS versions.

Level 4, Optimizing — Managed services

The lifecycle of each capability is managed in a standardized, automated way. Capabilities, features and updates are delivered continuously with no impact on users. Any large changes instigated by platform providers include migration plans for existing users with defined responsibilities and timelines.

Platform capability providers take on the brunt of responsibility for maintenance, but there is a clear contract — a "shared responsibility model" — describing the responsibilities of users, enabling both sides to operate mostly autonomously.

Characteristics

- A shared ownership model clearly defines who is responsible for platforms and their capabilities and what is expected of users.
- Teams script both the execution of the upgrade and any rollback strategies to keep risk and impact low.

Example Scenarios

- The users of virtual machines are not required to manage anything to do with version upgrades. Their only requirement is to have a stage in their delivery pipeline that contains a representative smoke test. They are then asked to declare their application as having lower risk tolerance so as to wait for a fully hardened upgrade or higher tolerance to become an early adopter. The virtual machine capability then manages the automated release of upgrades including rollbacks after either smoke test or canary release failures.

Measurement

What is the process for gathering and incorporating feedback and learning?

By reacting to explicit and implicit feedback from users, organizations can increase user satisfaction and ensure long-term platform sustainability. Organizations must balance innovation and meeting user demands to keep platform relevance. As technology and user preferences change, platforms that are agile and responsive to these changes will stand out. Regularly revisiting and refining the feedback mechanism can further optimize platform development and improve user engagement.

Level 1, Provisional — Ad hoc

Usage and satisfaction metrics are gathered in custom ways, if at all, for each platform and capability. Outcomes and measures of success are not consistently aligned across capabilities, and therefore corresponding insights are not gathered. User feedback and instrumentation of platform use may not be gathered, or if it is, it will be informal. Decisions are made based on anecdotal requirements and incomplete data.

Characteristics

- No experience or opinions about how to measure success of platforms
- Use familiar tools to gather common metrics with limited intent and forethought
- Reliance on small amounts of data
- Difficult to secure user participation — users believe their feedback isn't considered
- If surveys are used, the questions change between runs, negating the ability to track progress

Example Scenarios

- A platform tech lead wants to improve the collaboration with users by adding key topics to their next quarterly planning. They decide to run a survey on what users would like to see. The response is overwhelming, which is exciting, but also results in a difficulty organizing and responding to all of the ideas. While some ideas influence the quarterly planning, the users do not see their ideas as being accepted and are less inclined to reply to the next survey.
- The team wants to capture more data automatically, so they look for opportunities for easy collection such as test failures in CI. However, not every team uses the same CI automation so the data is only available for Java applications even though some teams have moved on to writing their services in Scala.

Level 2, Operationalized — Consistent collection

Organizations at this level have an intentional goal to verify platform products meet the needs of their market of internal users. Actionable, structured collection of user feedback is valued. Dedicated teams or individuals might be assigned to gather feedback, ensuring a more consistent approach. Feedback channels, such as surveys or user forums, are standardized, and feedback is categorized and prioritized. Beyond user feedback, there is also an expectation that user experiences are instrumented to generate usage data over time.

Challenges remain in translating feedback into actionable tasks. While there is a growing repository of user data, the organization might need help effectively understanding and integrating this feedback into a platform roadmap. It may be hard to ensure that users see tangible changes driven by their feedback.

Characteristics

- Data collection is discussed as part of most major planning sessions or capability implementations.
- There may not be alignment on exactly what to measure to verify success.
- Platform features can be measured for success, such as by measuring user adoption or user time saved.

Example Scenarios

- A platform team allocates 20% of their time to user defined features, which they identify based on surveys and other interview techniques. Their findings are collected into a tool that enables additional voting and commenting to further refine priorities. During implementation the requesting users are approached for collaboration on early designs and implementations. Once implemented, there are announcements which make sure requesting users are aware of new features and supported in adopting them.
- The team focused on software delivery capabilities wants to capture more data automatically including cycle time which they automate through the build tool from commit to production. There is an understanding that cycle time can include other activities like PR review, but that isn’t included at this time.

Level 3, Scalable — Insights

While robust, standard feedback mechanisms already exist, at this stage data is collected in crafted ways to yield specific strategic insights and actions. Desired results and outcomes are identified followed by standard metrics chosen to indicate progress towards those outcomes. Industry frameworks and standards may be used to benefit from industry research on the impact of certain behaviors.

Dedicated teams or tools are employed to gather and review feedback and summarize actionable insights. A symbiotic relationship between platform products and their users is established. Feedback is considered a strategic asset that guides platform operations and roadmap. Regular feedback review sessions might be instituted, where cross-functional teams come together to discuss and strategize based on user insights.

Characteristics

- Before delivering any new platform feature, the team discusses how to evaluate the outcome from their work.
- The organization has broad alignment on measures that indicate success of platform initiatives.
- A product manager or dedicated team member drives an ongoing and consistent feedback collection and analysis process.
- The organization has established metrics and goals to observe and target to indicate success.

Example Scenarios

- The organization has consistently tracked build times and lead time. However, now they realize that while easy to collect, these alone do not give a complete picture of software delivery. With this in mind, the team implements measurement for service reliability and stability.

Level 4, Optimizing — Quantitative and qualitative

Feedback and measurements are deeply integrated into the organization's culture. The entire organization, from top-level executives to engineers organization-wide, recognizes the value of data collection and feedback on product evolution. There is a democratization of data, where various stakeholders, including platform users and business leaders, are actively involved in identifying hypotheses for platform improvements, providing feedback during the design process, and then measuring the impact post delivery. All of these measurements are considered when planning platform initiatives.

Not only are standard frameworks leveraged, but there is an understanding that measuring from multiple angles creates a more holistic picture. There is an investment in understanding how qualitative measures change as quantitative ones are improved. There is a focus on identifying leading measures which can allow anticipation of features that would support user needs, alleviate their challenges, and stay ahead of industry trends and business requirements.

Characteristics

- Platform teams continuously seek ways to improve the metrics they watch and the way they gather data.
- The organization is familiar with and sensitive to Goodhart's Law: "When a measure becomes a target, it ceases to be a good measure."
- Metrics and telemetry gathered is continuously evaluated for true insight and value.
- Metric data management is well supported, such as standard platform capabilities to manage data lakes and derive insights.
- Cross-departmental collaboration is encouraged to avoid data silos and enable effective feedback cycles.

Example Scenarios

- Over time the organization has collected data indicating a rise in build time of over 15%. This triggers negative developer experiences and once triggered, even if the build time is reduced below the original time, developers stay frustrated for longer. This insight drives the build team to set and adhere to a Service Level Objective (SLO), which enables early identification and improvement before instigating the negative cycle with their users.

Conclusion

Platforms and their maintainers provide a foundation for agile digital product development. They provide a consistent collection of capabilities that enable efficient software development and delivery. This maturity model provides a map for your platform engineering journey.