Key Drivers/Stages	Initial	Repeatable	Defined	Managed	Optimizing
Investment How are staff and funds allocated to platform capabilities?	**Voluntary** Capabilities for common or critical functionality arise through necessity, largely by volunteers addressing their own current tactical user requirements. Leadership reacts to crises rather than proactively driving change, leading to fragmented collaboration and inefficiencies. **Budget and Staffing** Urgent requirements are addressed unfunded, on top of engineers current responsibilities. **Scope Management** Engineers focus on immediate needs within a specific context, sharing solutions minimally across wider contexts. **Measuring ROI** ROI is measured by how effectively the requirement was addressed and the impact on core project work.	**Ad-hoc Contributions** Dedicated teams are formed to address recurring challenges such as inconsistent infrastructure provisioning, fragmented security practices and pipeline bottlenecks, but efforts remain largely reactive. Leadership promotes basic collaboration and introduces metrics. **Budget and Staffing** Teams are created to work on key cross-cutting concerns, often reactively. Limited empowerment across the organization. **Scope Management** Scope is limited to specific concerns without a holistic approach to platform-wide improvement. **Measuring ROI** Improvements are measured in key cross-cutting concerns, such as the size of the backlog.	**Operationalized - Dedicated Team** Centrally funded teams—often called DevOps, Engineering Enablement, or Platform—focus on reactive technical requirements are treated as cost centers. Leadership fosters cross-functional teamwork and initial DevOps practices but struggles with measuring the platform team's value and aligning solutions with user needs. **Budget and Staffing** Central teams are funded based on existing technical requirements to accelerate software delivery. **Scope Management** Broad solutions addressing needs across teams. Short term focus on immediate returns rather than long-term growth. **Measuring ROI** ROI is demonstrated by measuring improvements in speed of delivery.	**Scalable - As Product** A cultural shift occurs where developers are treated as valued customers, with leadership emphasizing empathy, growth, a product-led approach, and continuous improvement. Funds and plans are allocated based on KPIs and feedback loops, enabling platform teams to optimize and make positive contributions to the business. **Budget and Staffing** Platform team is staffed and managed like product teams with dev, PM, UX and UXR roles. **Scope Management** Product roadmaps defined and reviewed with engineering teams set scope, and focus on meeting the needs of all development teams. **Measuring ROI** ROI is measured through improvements in developer satisfaction.	**Optimizing - Enabled Ecosystem** Leadership promotes innovation to maintain platform relevance. Platform teams optimize beyond basic capabilities. Specialists from domains like security and performance introduce advanced features, empowering product teams without relying on a centralized backlog. **Budget and Staffing** Central platform team encourages contributions across the organization; all teams have funding to contribute to the platform. **Scope Management** Engineers focus on enabling platform contributions for rapid knowledge sharing organization-wide. **Measuring ROI** ROI is measured through improvements in developer satisfaction.
Adoption Why and how do users discover and use your platform engineering solution and its capabilities?	**Informal** Adoption of shared platforms is sporadic and inconsistent, with no organization-wide strategy or guidance. Teams independently improve processes without coordination, lacking standardization across the organization. External tools are often considered more effective than internal ones. **Discover Services, Tools and Technologies** Tools are discovered informally, often through word-of-mouth or chance encounters. **Choose Services, Tools and Technologies** Engineering teams independently choose and integrate services based on their specific needs. **Use Services, Tools and Technologies** Teams maintain their own scripts and tools unique to their specific contexts and needs.	**Mandated** The organization mandates shared platforms through internal directives, requiring their use for some cases. Capabilities cover typical use cases but not outliers, making it hard to add unusual requirements to the platform. User discovery is inconsistent; often reliant on platform team guidance. **Discover Services, Tools and Technologies** Teams must seek platform team guidance, provided via internal documentation or directives. **Choose Services, Tools and Technologies** Engineering teams rely on informal discussions with the platform team to choose mandated services if they meet their needs. **Use Services, Tools and Technologies** Processes built around platform standards but not easily extended; teams may fail to use them or be unsatisfied with the results.	**Advertised** Platform capabilities are actively promoted, communicating benefits that align with team needs. The platform team collaborates with engineering teams, offering high-quality services to reduce operational overhead. Some teams still perceive low ROI in migrating to the platform due to established practices and technical debt. **Discover Services, Tools and Technologies** Engineering teams discover platform capabilities through platform team directives covering typical use cases. **Choose Services, Tools and Technologies** The platform team collaborates with engineering teams to encourage the use of platform capabilities. **Use Services, Tools and Technologies** Problems and solutions are shared through informal communities; ambassadors within teams advocate for platform use.	**Value-Driven** Product teams choose platform capabilities due to clear value in reducing cognitive load and providing high-quality services. Documentation and ergonomic interfaces enable quick provisioning, leading teams to prefer internal platforms over developing capabilities themselves or hiring providers. **Discover Services, Tools and Technologies** Engineering teams actively engage with the platform via self-service UX to discover capabilities. **Choose Services, Tools and Technologies** Teams look to the platform for technical solutions, guided by the value described for each capability. **Use Services, Tools and Technologies** Platform use is fully supported through templates, support forums, and documentation; teams apply templates and engage in forums.	**Participatory** Product team users invest further by contributing back to platform capabilities, improving existing ones and adding new features. Processes enable users to identify requirements and coordinate contributions, with new capabilities published consistently and fully documented. **Discover Services, Tools and Technologies** Developer advocates and ambassadors build an internal user community, extending platform ownership to contributors. **Choose Services, Tools and Technologies** Platform engineers participate in product team planning to learn requirements and suggest capabilities. **Use Services, Tools and Technologies** Engineering teams are empowered to contribute fixes and features, generating pull requests and participating in reviews.
Governance How do you ensure that your users have access to the resources and capabilities that they need and that costs, data and IP are appropriately managed?	**Independent** The organization relies on manual processes for compliance, enforced through centralized control and gatekeeping, hindering scalability and sustainability in governance. Developers and security teams operate independently with minimal collaboration, leading to reactive responses to policy violations. **Define Policies and Frameworks** A central governance team defines security and compliance measures individually for each team or project. **Implement Policies** Compliance meets essential standards without formal processes; security measures like secret management added as afterthoughts. **Monitor and Mitigate** Incidents are addressed after they occur, with no formal processes to prevent policy violations or security breaches. **Manage Access** Permissions are granted based on immediate needs.	**Documented** The organization begins to document and share security and compliance policies across teams, but these remain basic and unevenly applied. Centralized systems like ticketing manage policy reviews, introducing bottlenecks due to manual audits. Improvements in traceability are limited by lack of uniformity and enforcement. **Define Policies and Frameworks** Common tools are introduced, but policy creation is still largely manual and lacks uniformity; policies are rudimentary. **Implement Policies** A central governance team manually applies policies during key stages, with some efforts to standardize integration across teams. **Monitor and Mitigate** Basic auditing processes are established for some key areas. By nature these are reactive. **Manage Access** Some standard roles and permissions are established but not universally; processes are documented but not enforced uniformly.	**Standardized** The organization centralizes governance to reduce variability and improve efficiency, introducing standardized processes for consistent security and compliance across all teams. Implementing these practices requires significant coordination and expertise, and development teams have limited ability to change policies. **Define Policies and Frameworks** Policies are standardized and centrally managed with established documentation and control mechanisms. **Implement Policies** Policy implementation is centrally managed with some automation via a review and/or ticketing process. **Monitor and Mitigate** Key governance standards upheld through systematic application of monitoring processes, regular auditing of all platform activities. **Manage Access** Access control is centralized and automated, with a formal RBAC system defining roles and permissions aligned with job functions.	**Integrated** Security and compliance are fully integrated into workflows, using automation to consistently apply and update policies across systems and teams. The focus shifts to actively preventing gaps and overlaps in governance, deploying advanced tools and real-time analytics for monitoring. **Define Policies and Frameworks** Policies are regularly reviewed and refined based on feedback and operational needs. **Implement Policies** Policies are integrated into reusable templates and workflows and embedded into CI/CD pipelines, enabling automated checks. **Monitor and Mitigate** Advanced tools and analytics monitor platform activities in real-time, enabling quick detection and response to threats and violations. **Manage Access** Policies enforce least privilege with automated access reviews; a comprehensive IAM system aligns access rights with org changes.	**Predictive** The organization adapts and responds to changing conditions, balancing centralized control with adaptive, context-aware access management. Predefined compliance options ensure organizational standards are met, enabling efficient and tailored workflows within regulatory requirements. **Define Policies and Frameworks** Policies are continuously refined and optimized based on advanced analytics and predictive feedback. **Implement Policies** "Get right" campaigns are launched to ensure existing applications align with current best practices. **Monitor and Mitigate** Predictive analytics identify potential threats before they materialize, allowing proactive risk mitigation. **Manage Access** Context-aware access control dynamically adjusts permissions based on real-time factors like location, and time of access.
Provisioning and Management How do your users create, deploy and manage resources?	**Manual** Developers manually set up infrastructure based on guidance from IT or Architecture teams. This leads to inconsistencies and delays, as provisioning requests are manually reviewed, increasing the risk of errors. Siloed operations and a lack of formal processes create inefficiencies, making this approach unsustainable as demands grow. **Define Provisioning Processes** Developers manually set up infrastructure on a case-by-case basis, often using varied methods. **Respond to and Manage Requests** Infrastructure requests are handled manually through informal communication, with manual IT reviews. **Monitor Resource Allocation** Teams rely on manual checks, often responding only when capacity issues arise.	**Coordinated** The organization takes initial steps toward centralizing and standardizing provisioning processes. Centralized ticketing systems manage infrastructure requests, and some errors are removed but manual approvals still introduce bottlenecks, slowing down development and deployment cycles. **Define Provisioning Processes** Some teams start using similar tools and scripts, but there is no organization-wide standard. **Respond to and Manage Requests** Requests are submitted through a centralized system, with manual approval and provisioning by IT or platform teams. **Monitor Resource Allocation** Teams start using standard tools for resource monitoring, but the view remains siloed and project-specific.	**Paved** Provisioning processes are formalized across the organization, using Infrastructure as Code (IaC) practices with consistent templates and tools. However, the platform team may struggle to keep up with the growing volume of requests, highlighting the need for scalable solutions. **Define Provisioning Processes** IaC templates standardize provisioning processes across all teams. **Respond to and Manage Requests** Requests are handled through structured workflows using predefined templates and processes (e.g., paved paths). **Monitor Resource Allocation** Centralized dashboards track resource allocation across services, offering performance insights.	**Automated** Provisioning is automated and integrated into CI/CD pipelines, reducing manual effort and ensuring consistent deployments. Governance and compliance checks are embedded into workflows, but the shift to "everything as code" requires expertise in automation and coding. **Define Provisioning Processes** Provisioning is fully automated within CI/CD pipelines, including governance and compliance checks. **Respond to and Manage Requests** An automated self-service capability allows users to provision resources independently within controlled parameters. **Monitor Resource Allocation** Automated scaling adjusts resource allocation based on usage patterns to ensure optimal performance.	**Adaptive** Provisioning becomes adaptive, with intelligent systems anticipating infrastructure needs in real-time. This approach balances flexibility and governance, ensuring resources are allocated efficiently while maintaining compliance. The challenge lies in managing centralized governance without stifling innovation. **Define Provisioning Processes** Automated provisioning integrated into CI/CD pipelines adapts based on real-time needs, maintaining governance and compliance. **Respond to and Manage Requests** Intelligent systems handle requests proactively, anticipating needs while balancing flexibility and governance. **Monitor Resource Allocation** Predict and respond to resource demand based on historical data and current usage, optimizing performance and cost-efficiency.
Interfaces How do users interact with and consume platform capabilities?	**Custom Processes** A variety of inconsistent processes exist to provision capabilities. Custom, manual processes address immediate team needs, with knowledge shared informally. Provisioning services often require deep support from providers due to a lack of standardization. This approach prioritizes speed over standardization. **Setup Environment** Individual engineers independently set up environments by consulting colleagues or using personal practices. **Diagnose Application Behavior** Engineers choose their tools and processes for diagnosing application behavior, managing their access to applications and logs.	**Local Standards** Engineers informally define standards to enhance knowledge sharing, but reliance on individual commitment makes consistency challenging. Support communities form around these standards, but they depend on resources from specific teams. Over time, standards diverge, requiring effort to reconcile. **Setup Environment** Teams define their own tools and processes, possibly using documentation or containers, driven by team preferences. **Diagnose Application Behavior** Teams establish their own practices for diagnosing behavior, often relying on DevOps or IT teams for access to deployed resources.	**Standard Tooling** Consistent interfaces for provisioning and observing platforms meet broad needs. "Paved roads" or "golden paths" (documentation and templates) define how to provision capabilities using compliant patterns, though expert support is still often needed. Central teams manage templates and documentation, requiring significant effort. **Setup Environment** Required tools and processes defined in docs and templates. Teams may modify templates but may not merge changes back centrally. **Diagnose Application Behavior** Standard practices are defined for accessing and diagnosing deployed resources.	**Self Service Solutions** Solutions offer user autonomy with minimal maintainer support, providing consistent interfaces for easy discoverability and portability across capabilities. A guided internal language helps users adopt and integrate platform capabilities quickly, creating a user-centric, self-serviceable, and consistent collection of capabilities. **Setup Environment** The platform aids knowledge sharing through discovery and modification of templates, increasing its value. **Diagnose Application Behavior** Tools and services for observing resources are provided on demand through the platform.	**Integrated Services** Platform capabilities are integrated into the tools and processes teams already use, with some capabilities automatically provisioned. Platform capabilities provided as building blocks for flexible, higher-level use cases requiring deeper customization. Platform teams assess which capabilities are effective to guide further investments. **Setup Environment** Platform capabilities are integrated into tools teams already use (CLI, IDE), fitting naturally into their workflows. **Diagnose Application Behavior** The platform automatically sets up observability for deployed applications, providing access to diagnostic data and applications.
Measurements and Feedback What is your organization's process for gathering and incorporating metrics and feedback and how do you measure success of your platform engineering practices?	**Ad-hoc** Measurements are gathered inconsistently without alignment to organizational goals, leading to fragmented outcomes. User feedback is informal and anecdotal, resulting in decisions based on incomplete data. Volunteer driven efforts measured against local requirements, making true ROI assessment difficult. **Measure Success** ROI measurement is fragmented and relies on basic activity-based metrics, offering limited insight into broader goals. **Feedback Collection** Feedback is collected informally, often in response to specific issues. **Analyze Feedback** Feedback is rarely analyzed in-depth, leading to decisions based on incomplete data. **Engage Stakeholders** Minimal engagement; feedback is gathered from a limited user group. **Document Learnings** Little to no documentation of feedback processes or outcomes.	**Structured Processes** Basic feedback mechanisms like surveys or forums are established, capturing user experiences more systematically, but processes vary between teams. Comparisons of costs to benefits like reduced backlogs are limited and don't reflect true business value. Feedback remains disconnected from the broader platform strategy. **Measure Success** Measurements focus on process efficiency metrics like deployments or timelines, offering some insight but still activity-focused. **Feedback Collection** Basic channels are established, but consistency varies between teams, focusing on general performance metrics. **Analyze Feedback** Feedback remains largely informal and bottom up. Although it begins to influence planning it lacks alignment with overall strategy. **Engage Stakeholders** Dedicated individuals start managing feedback, involving more users but on a limited scale. **Document Learnings** Initial documentation is created but isn't comprehensive or consistently used.	**Consistent** Structured feedback provides clearer insight into organizational impact. Costs are compared with tangible benefits, leading to a more meaningful ROI assessment. Challenges remain in translating feedback into actionable tasks and balancing short-term efficiency with long-term value. **Measure Success** Focus shifts to outcome-based metrics like developer productivity. Linking metrics to financial performance remains challenging. **Feedback Collection** Feedback collection is formalized and standardized, focusing on a deeper understanding of user needs alongside key metrics. **Analyze Feedback** Systematic analysis using qualitative and quantitative methods; standard metrics like DORA or SPACE are employed. **Engage Stakeholders** Regular review sessions are held with cross-functional teams; stakeholders actively interpret feedback and make decisions. **Document Learnings** Comprehensive documentation of processes, outcomes, and lessons learned is maintained and shared.	**Insights** Organizations have robust feedback mechanisms and measurements focused on strategic business outcomes. Data guides platform operations and feedback is integrated into platform roadmaps. Challenges remain in refining measurement frameworks to consistently and accurately capture value-centric metrics. **Measure Success** The organization employs advanced analytics to assess the platform's impact on outcomes like revenue growth. **Feedback Collection** Integrated feedback mechanisms are embedded into development and operations to identify key improvement areas. **Analyze Feedback** Feedback is correlated with performance metrics to drive strategic improvements. **Engage Stakeholders** Stakeholders across the organization are deeply involved in the feedback process; collaboration is structured to avoid silos. **Document Learnings** Dynamic, real-time documentation is maintained, reflecting ongoing feedback and lessons learned, and is accessible to all stakeholders.	**Proactive** Feedback and measurements are deeply integrated into the organization's culture, driving the platform's ability to anticipate and adapt to future challenges and opportunities. Data is democratized, involving platform users and business leaders in identifying hypotheses for platform improvements. **Measure Success** The organization leverages predictive analytics and advanced metrics to anticipate future needs and opportunities. **Feedback Collection** Integrated feedback mechanisms are embedded into development and operations to identify key improvement areas. **Analyze Feedback** Feedback is integrated into a continuous improvement cycle. **Engage Stakeholders** A culture of feedback is established across the organization; feedback is democratized with input from all levels. **Document Learnings** Dynamic, real-time documentation is maintained, reflecting ongoing feedback and lessons learned, accessible to all stakeholders.