Exercise 2.2: Define the feature set of an MVP engineering platform

Create a detailed list of the capabilities you believe are necessary to form the MVP for an engineering platform. For each, describe why the absence of the capability renders the MVP definition of the product unsuccessful.

Solution to Exercise 2.2: Feature set of an MVP engineering platform

A proposed list of capabilities for an engineering platform's initial, minimum valuable product definition.

1. **Identity**. Programmatic access to an authoritative source for authentication and authorization. It must be able to effectively integrate with the PE product roadmap for creating team-level tenancy and resilient management technology and tool integrations.

   Discuss: This is the most basic access and security requirement. Any solution that is not sufficiently programmatically manageable by the Platform team to create a product experience will impact every feature in the roadmap to some degree. In other words, identity is a dependency for every other capability of the platform. Getting this right first will create more flexibility in prioritizing other capabilities than any other architectural decision.

2. **General Developer Tools**: source version control, secrets management, artifact stores that support OCI images and related metadata, CI and CD pipeline orchestration.

   Discuss: These tools are likely already available in some fashion. Where the current tool is a good long-term choice for integration with the platform, the timing of that integration may be flexible. Even if the current solution is inadequate, a new tool or integration may not technically be needed if users can continue with their existing solution without modification. In either case, programmatic access to these tools is a minimal requirement for the Platform product team to provide a successful self-service experience.

3. **Public or private cloud**. Regardless of the infrastructure capabilities that make up the MVP, we must begin with a target location where the platform engineers have programmatic access to automate the lifecycle of infrastructure resources as a self-serve experience for users.

   Discuss: The underlying network is frequently the biggest source of friction at the start. If an architecture that enables the Platform team to self-manage networking and IP needs can not be agreed upon, then minimally provide enough capacity and connectivity to cover the expected scale for the first 18-24 months after users are on the platform. Shortcuts here become ticking time bombs for later problems, even outages.

4. **Dynamic compute**. The single most frequent behavior for the customers of the platform will be deploying and interacting with the software they are building. As most new strategic custom software is non-mobile and follows a stateless or distributed service architecture, the most needed type of compute will be Kubernetes orchestration. This also meets our MVP need for a control plane to manage the authentication and authorization experience defined through the resources provided in #1.

   Discuss: However, we will not just assume containers are the most needed thing; we will investigate our internal customer pool in case a different architecture has a broader need.

5. **Ingress and traffic routing**. DNS and load balancing of some form is required. It is possible that there is a sufficiently large pool of users building software that is only accessed from other internal systems. In that case, the MVP does not need to solve for external access. Though Kubernetes provides a category of traffic routing, this capability also covers whatever is minimally necessary for traffic between the developer tools and the internal or external traffic to the clusters themselves. Given how extensively a service mesh impacts the effectiveness of any traffic routing requirements and the potential cost of later adoption, a

mesh should be included in the MVP though using mesh features can be limited to ingress. If external access is necessary, this covers things like firewalls and other required edge capabilities.

Discuss: Software that can't be accessed provides no value.

6. **Observability tools**. Aggregated logs, metrics, and events from all of the MVP technologies, along with a self-serve means of searching and filtering the aggregated data and a software-defined means of managing dashboards, monitoring, and alerting.

7. **Software security and provenance**. This refers to the security activities of the CI and CD pipelines or other security services directed at the software platform users deploy. If the platform won't initially provide language starter kits that include pipelines because existing pipelines can be used to start, and those pipelines include the basics, such as vulnerability scans, then you could consider leaving this out of the MVP. Given the potentially bankrupting impact of security vulnerabilities, the platform MVP should at least include confirming provenance.

   Discuss: The secure configuration of the platform's components is generally assumed to be part of each component implementation, whether the technology is included in the MVP or not.