Internal Developer Platform Documentation

1. Introduction

Overview of the Internal Developer Platform (IDP)

- **Purpose and Benefits:** The IDP is designed to streamline and accelerate the software development process within the organization. It provides a centralized platform for developers to access tools, resources, and services, fostering collaboration and efficiency.
- **Key Components:**
    - **Unified Toolset:** The IDP integrates a wide range of development tools, including version control systems, build automation tools, and testing frameworks, providing developers with a consistent and efficient workflow.
    - **Deployment Automation:** Automated pipelines for building, testing, and deploying applications, reducing manual effort and minimizing the risk of errors.
    - **Monitoring and Analytics:** Tools for monitoring application performance and collecting usage data, providing insights for optimization and troubleshooting.
- **Self-Service Capabilities:** The IDP empowers developers with self-service capabilities, allowing them to provision resources, manage environments, and deploy applications independently, without relying on manual intervention from operations teams.
- **Governance and Security:** The IDP incorporates robust governance and security controls to ensure compliance with organizational policies and protect sensitive data.
- **Collaboration and Knowledge Sharing:** The IDP fosters a culture of collaboration and knowledge sharing among developers through features such as wikis, forums, and chat channels.

2. Onboarding Requirements

- **Internal Billing ID:**
    - The internal billing ID is used to track and manage financial records for accurate billing and revenue reporting.
    - Here is a list of the appropriate internal billing ID's by department , please identify the one that corresponds to your application and use it during the onboarding process :

Department Name	Billing ID
Engineering	DEPT-ENG-1001
Product Management	DEPT-PRD-2053
Marketing	DEPT-MKT-3117
Sales	DEPT-SAL-4890
Human Resources	DEPT-HRS-5024

Finance	DEPT-FIN-6731
Information Technology	DEPT-ITS-7448
Customer Support	DEPT-SUP-8192
Research & Development	DEPT-RND-9005
Legal	DEPT-LEG-1567

Example/Test Billing ID:

For testing purposes or onboarding the sample application provided with the platform documentation, you can use the following billing ID:

- **Test Billing ID:** `DEPT-SAMPLE-12345`

- **Application Registration (Internal CMDB):**

  Registering applications within the CMDB is essential for maintaining an accurate, up-to-date repository of configuration items (CIs) and their relationships. This centralized source of truth enables numerous critical IT management functions:

- **Incident and Problem Management**: Quickly identify impacted applications during outages, streamlining troubleshooting and root cause analysis.
- **Change Management**: Assess potential ripple effects of changes, ensuring that modifications are implemented safely and with minimal disruption.
- **Asset Management**: Track application lifecycles, from procurement to retirement, facilitating license compliance and cost optimization.
- **Capacity Planning**: Forecast resource requirements based on application dependencies and usage patterns, preventing performance bottlenecks.
- **Service Level Management**: Establish and monitor SLAs based on accurate application information, ensuring that business needs are met.
- **Compliance and Auditing**: Demonstrate regulatory compliance by maintaining a comprehensive inventory of applications and their configurations.
- **Security Management**: Identify and mitigate vulnerabilities by understanding application interdependencies and access controls.

  By maintaining an accurate and comprehensive CMDB, organizations can improve operational efficiency, reduce downtime, enhance security, and ensure that IT services align with business objectives.

Registering your application in the company's internal Configuration Management Database (CMDB) is a mandatory step before onboarding it to the Internal Developer Platform. This ensures your application is properly tracked for asset management, incident response, change management, and compliance purposes.

Steps to Register your App in the CMDB:

1. **Access the CMDB Portal:** Navigate to the internal CMDB registration site, typically found at: `http://your-internal-cmdb.company.com` (Note: Replace with your actual internal URL). You may need to log in with your standard company credentials.
2. **Initiate New Registration:** Look for an option like "Register New Application," "New Configuration Item," or similar.
3. **Complete Registration Form:** Fill out the required information about your application. You will likely need to answer questions such as:
    - **Application Name:** The official, user-facing name of your application.
    - **Brief Description:** A short summary of the application's purpose and functionality.
    - **Business Owner / Sponsoring Team:** The primary business contact or team responsible for the application.
    - **Technical Owner / Development Team:** The primary technical contact or team responsible for development and maintenance.
    - **Application Criticality:** The level of impact to the business if the application were unavailable (e.g., High, Medium, Low).
    - **Data Sensitivity:** Classification of the type of data the application handles (e.g., Public, Internal, Confidential, Restricted).
4. **Submit for Approval:** Once the form is complete, submit it for registration. Depending on the process, this might involve an approval step.
5. **Receive Application Identifier:** Upon successful registration and approval, the CMDB system will generate a **unique Application Identifier** (sometimes called a CMDB ID or Asset Tag). This identifier is crucial for linking your application across various internal systems.
6. **Record the Identifier: Carefully record this Application Identifier.** You will need to provide it during the onboarding process for the Internal Developer Platform to associate your deployed resources correctly.

Example/Test Identifier:

For testing purposes or onboarding the sample application provided with the platform documentation, you can use the following identifier:

- **Test Application Identifier:** `APP-SAMPLE-12345`

Prerequisites:

  - Required access (e.g., GitHub, Google Cloud).
  - Any necessary local setup (e.g., Git, Docker Desktop).

3. Golden Paths: Architecture & Usage

- The "golden paths" represent the officially supported, recommended, and optimized ways to build and deploy applications using this Internal Developer Platform (IDP). They are designed to provide a streamlined, efficient, and reliable experience by leveraging pre-configured tooling, automation (like the CI/CD pipelines triggered via GitHub), and infrastructure patterns managed by the platform team. Following a golden path ensures easier onboarding, better maintainability, standardized monitoring, and quicker access to support. While other approaches might be possible, deviating from a golden path may require significantly more effort from development teams to configure, manage, and maintain their application's infrastructure and deployment processes.

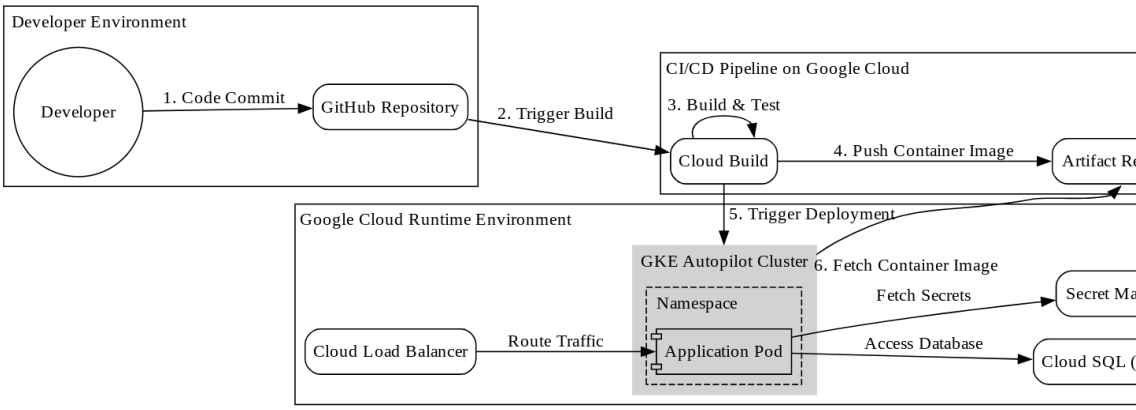
  The platform utilizes GitHub as the central repository for all application source code. As part of the onboarding process for a new project or application, the platform team will provide you with a dedicated GitHub repository. This repository comes pre-configured with a Cloud Build trigger specifically linked to it. Once you commit your code changes to the `main` branch of this provided repository, the associated Cloud Build pipeline will automatically kick off, initiating the build, test, and deployment process defined within your chosen golden path.

  Currently, there are two primary golden paths available:

3.1. Golden Path 1: Containerized Applications on GKE Autopilot

  - **Overview:** This path is the standard and recommended approach for most new development. It is ideal for applications that are already containerized or can be easily packaged into Open Container Initiative (OCI) containers. GKE Autopilot abstracts away much of the underlying infrastructure management, allowing teams to focus primarily on their application code.

- **Suitable Application Examples:**
  - Stateless web applications and APIs.
  - Microservices architectures where individual services are deployed as separate unprivileged containers.
  - Background processing workers or queue consumers.
  - Applications that benefit from automatic scaling based on load.
- **Architecture:**
  - ***Need a diagram of the GCP Components***
  - Key Components: GitHub, Cloud Build, Artifact Registry, GKE Autopilot, Cloud SQL (Postgres), Secret Manager, Cloud Load Balancing, IAM.
- **Workflow:**
  - Steps:
    1. Code commit to GitHub
    2. Cloud Build trigger initiated
    3. Cloud Build performs build and test routines
    4. Container image pushed to Artifact Registry
    5. Deployment triggered to GKE Autopilot
    6. Cluster pulls image from Artifact Registry

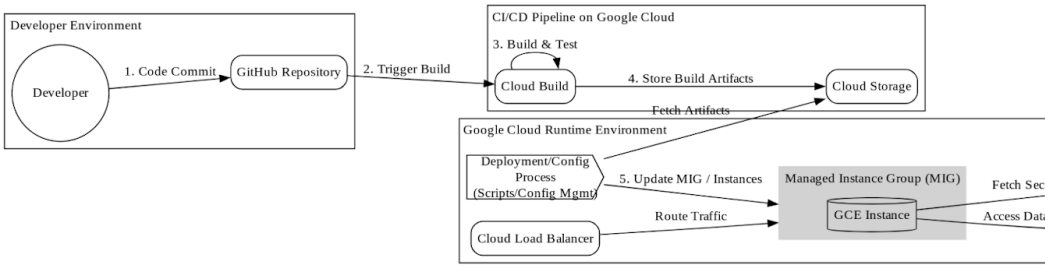


Golden Path 1: High Level GKE Autopilot Workflow

- **Getting Started Guide:** ***Need a Step-by-step tutorial for deploying a sample containerized application.***

- **3.2. Golden Path 2: Applications on Google Compute Engine (GCE) Instances**

    - **Overview:** This path provides more direct control over the operating system and underlying virtual machine environment. It's suitable for applications that have specific OS-level dependencies, require configurations not easily achievable within containers, or cannot be easily containerized. While offering flexibility, this path generally involves more infrastructure management responsibility for the development team compared to the GKE Autopilot path.
    - **Suitable Application Examples:**
        - Legacy, monolithic applications that are difficult or costly to containerize.
        - Applications requiring specific OS packages, kernel modules, or hardware configurations only available on GCE.
        - Certain types of stateful applications where managing state via external services like Cloud SQL is not feasible and instance-level state is preferred (use with caution).
        - Specialized third-party software requiring installation directly onto a virtual machine.
    - **Architecture:**
        - *(Diagram/Image illustrating the flow needed here)*
        - Key Components: GitHub, Cloud Build, Cloud Storage (for artifacts), Compute Engine (GCE), Cloud SQL (Postgres), Secret Manager, Cloud Load Balancing
    - **Workflow:**
        - Steps:
            1. Code commit to GitHub
            2. Cloud Build trigger initiated
            3. Cloud Build performs build and test routines
            4. Build artifacts stored in Cloud Storage
            5. Deployment/Configuration process updates GCE instances



Golden Path 2: GCE Instances Workflow (MIG with lhead)

Updates to GCE instances within the Managed Instance Group (MIG) for this golden path are handled using **GCE Startup Scripts**. The Instance Template configured for the MIG includes a startup script designed to prepare the instance and run the application.

Here's how the update process works:

1. **Artifact Storage:** As part of the CI/CD process (Step 4), Cloud Build places the latest build artifacts (e.g., application binaries, configuration files) into a designated location within Cloud Storage.
2. **Startup Script Execution:** When a new GCE instance is created within the MIG (either initially or during an update/scaling event), the defined startup script automatically executes.
3. **Fetching Artifacts:** The startup script contains logic to securely download the latest application artifacts from the specific Cloud Storage location where Cloud Build placed them.
4. **Configuration & Startup:** After fetching the artifacts, the script performs any necessary setup, such as installing dependencies, applying configurations, and finally, starting the application service.
5. **Triggering Updates:** To deploy a new version, an update is triggered on the Managed Instance Group (MIG) itself. This involves initiating a rolling replacement of the instances. As old instances are terminated and new ones are created based on the existing instance template, these new instances automatically run

the startup script, pulling the latest artifacts and configurations you deployed to Cloud Storage.

- **Getting Started Guide:** ***Need a Step-by-step tutorial for deploying a sample application onto a GCE instance.***

4. Onboarding to the Platform

TODO

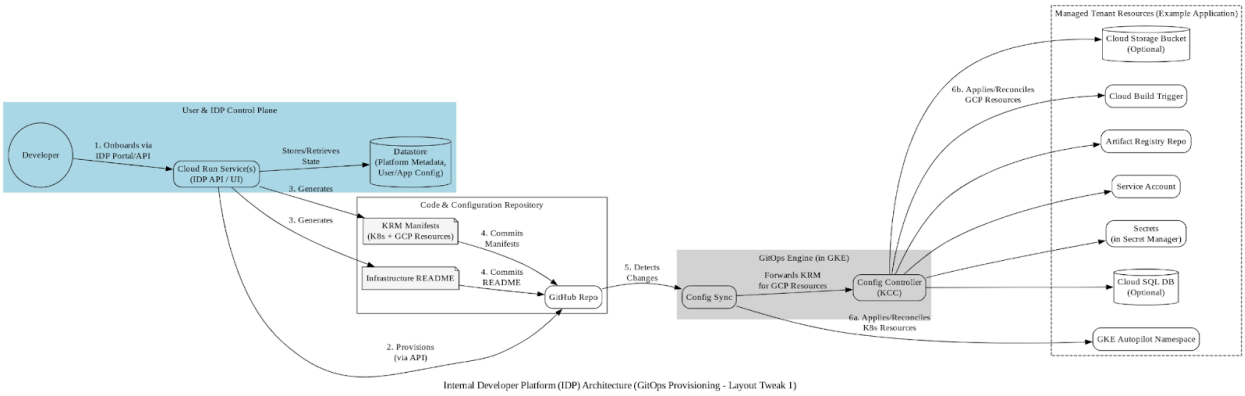
5. Platform Deep Dive

5.1 Services Available

- **GKE Autopilot:** This service is the foundation of our platform, managing and orchestrating containerized applications. It automates many operational tasks, such as scaling, node provisioning, and maintenance, allowing developers to focus on application development.
- **Cloud Build:** This service is our CI/CD pipeline, automating the building, testing, and deployment of our applications. It integrates with our source code repository and GKE Autopilot to enable fast and reliable software delivery.
- **Cloud Storage:** This service provides scalable and durable object storage for our platform. It stores application data, backups, and other artifacts including code deployment artifacts.
- **Cloud SQL (Postgres):** This service is our managed relational database. It stores structured application data and provides high availability, scalability, and security.
- **Artifact Registry:** This service stores and manages container images and other build artifacts. It integrates with Cloud Build and GKE Autopilot to enable efficient and secure image management.
- **Compute Engine:** This service provides virtual machines that can be used for various purposes, such as running batch jobs or hosting custom applications.
- **IAM:** This service manages access control and permissions for our platform. It ensures that only authorized users and services can access our resources.
- **Cloud Load Balancing:** This service distributes incoming traffic across multiple instances of our application, ensuring high availability and performance.
- **Secret Manager:** This service stores and manages sensitive data, such as API keys and database passwords. It integrates with our other services to provide secure access to secrets.

5.2 Platform Architecture

This section provides a high-level overview of the Internal Developer Platform (IDP) architecture itself – the components that run the platform and orchestrate the developer experience.



Internal Developer Platform (IDP) Architecture (GitOps Provisioning - Layout Tweak 1)

The Internal Developer Platform utilizes a hybrid approach, primarily leveraging a GitOps workflow facilitated by Google Cloud's Config Sync and Config Controller running within the GKE Autopilot cluster, but retaining direct API interaction for external resources like GitHub.

Here's how the provisioning process works with this model:

1. **Onboarding Request & Initial GitHub Repo Creation:** When a developer onboards an application via the IDP portal/API, the IDP Control Plane (Cloud Run service) first interacts *directly* with the GitHub API (using appropriate credentials/libraries) to create the dedicated source code repository for the application.
2. **Manifest Generation:** Once the repository is created, the IDP Control Plane generates a set of declarative configuration manifests. These manifests adhere to the Kubernetes Resource Model (KRM) and define:
    - Standard Kubernetes resources needed within the application's namespace (e.g., NetworkPolicies, ResourceQuotas).
    - Google Cloud resources required by the application, expressed as KRM objects (e.g., `SQLInstance`, `ServiceAccount`, `IAMPolicyMember`, `StorageBucket`, `ArtifactRegistryRepository`, `CloudBuildTrigger`). *(Note: Cloud Build Triggers can often be defined via KRM)*.

    - Initial application deployment manifests (e.g., Kubernetes Deployment, Service) and a baseline `README.md`.
3. **Commit to Git:** The IDP Control Plane commits these generated manifests to the newly created GitHub repository (or potentially a central configuration repository monitored by the platform). This Git commit becomes the desired state declaration for the application's Google Cloud infrastructure, Kubernetes configuration, and base application setup.
4. **Config Sync Detection:** Config Sync, running in the GKE cluster and configured to watch the relevant repository path, automatically detects the new or changed manifests committed by the IDP.
5. **Applying K8s Resources:** Config Sync directly applies any standard Kubernetes resource manifests (like Namespaces, NetworkPolicies) to the GKE cluster.
6. **Config Controller Reconciliation:** Config Sync forwards the KRM manifests for Google Cloud resources (like `SQLInstance`, `ServiceAccount`, `CloudBuildTrigger` etc.) to Config Controller.
7. **GCP Resource Provisioning via Config Controller:** Config Controller acts as a Kubernetes operator. It reads the desired state from the KRM manifests and makes the necessary calls to the corresponding Google Cloud APIs (e.g., Cloud SQL API, IAM API, Cloud Build API) to create or modify the actual cloud resources, continuously working to reconcile the live state with the desired state defined in Git.

In this model, the initial setup of external resources like the GitHub repository requires direct API calls from the IDP Control Plane. However, the subsequent provisioning and management of Google Cloud resources and Kubernetes configurations are handled declaratively via manifests in Git, reconciled by Config Sync and Config Controller.

5.3 IDP Control Plane

The core of the IDP operates as a control plane hosted on Google Cloud. This control plane is responsible for managing platform state, handling user interactions, and automating resource provisioning. The key components are:

- Cloud Run Service(s) (IDP API / UI): One or more serverless Cloud Run services provide the main interface to the IDP. This includes the web portal developers use for

onboarding and self-service actions, as well as the backend APIs that orchestrate the platform's functions.
- Datastore (Platform Metadata): Google Cloud Datastore is used to store essential platform metadata. This includes information about onboarded users, registered applications, provisioned resource details, and other configuration required for the IDP to operate. The Cloud Run services interact with Datastore to store and retrieve this state information.

5.4 Developer Onboarding and Resource Provisioning

Developers interact with the IDP primarily through the API or UI hosted on Cloud Run. The typical workflow involves:

1. **Onboarding Request**: A developer initiates the onboarding process for a new application via the IDP portal or API, providing necessary details like the Application Identifier (obtained from CMDB registration) and the desired Golden Path.
2. **Automated Provisioning**: Upon successful onboarding validation, the IDP Control Plane automatically provisions the necessary infrastructure and configuration for the application based on the chosen golden path. Common resources provisioned include:
    - GitHub Repo: Creating a dedicated repository in GitHub for the application's source code.
    - Cloud Build Trigger: Configuring a trigger linked to the GitHub repo to automatically start the CI/CD pipeline on commits to the main branch.
    - Service Account: Creating a dedicated Google Cloud Service Account with appropriate permissions for the application to interact with other Google Cloud services.
    - Secrets: Provisioning necessary secrets (e.g., database credentials, API keys) securely within Google Secret Manager.
    - Cloud SQL DB / Cloud Storage Bucket (Optional): Provisioning a Cloud SQL database instance or a Cloud Storage bucket if requested during onboarding or required by the golden path.
3. **Specific resources are provisioned based on the chosen path**:
    - For Golden Path 1 (GKE Autopilot):
        - An Artifact Registry Repo is set up to store the application's container images.
        - A dedicated GKE Autopilot Namespace is allocated within the shared cluster for the application's workloads.
    - For Golden Path 2 (GCE Instances):
        - A GCE Instance Template is configured, including the necessary startup

scripts.
    - A Managed Instance Group (MIG) is created based on the instance template to manage the application's VMs.
    - Cloud Storage is used by the Cloud Build pipeline specifically to store build artifacts (e.g., application binaries, deployment scripts) that will be fetched by the GCE instances during startup.

4. **Code Scaffolding and Configuration: As part of the setup, the IDP may also:**

    - Generate Files: Create initial configuration files like baseline Kubernetes manifests (for Path 1) or deployment scripts.
    - Generate Documentation: Create a basic `README.md` file outlining the provisioned infrastructure.
    - Push to Repo: Commit these generated files to the newly provisioned GitHub repository to provide a starting point for the developer.

6. Support & Contact

If you encounter issues, have questions, or need assistance with the Internal Developer Platform (IDP), here are the primary ways to get support:

- **Slack Channel (for quick questions & discussion):**

    - Join the **`#idp-support`** channel in Slack (Note: Replace with your actual chat channel details)
    - This is the best place for general questions, quick troubleshooting help, sharing tips with other users, and staying updated on platform announcements. The platform team monitors this channel during business hours.
- **Ticketing System (for bug reports & feature requests):**

    - For reporting bugs, requesting new features, or tracking issues that require more in-depth investigation, please file a ticket in the company's standard ticketing system (e.g., Jira, ServiceNow).
    - Be sure to assign the ticket to the **`IDP Support`** queue or component (Note: Replace with your actual team support queue/bug component)
    - Please include detailed information, such as steps to reproduce the issue, error messages, relevant application identifiers, and the golden path you are using.

For direct inquiries to the platform team, especially for questions not suitable for the public Slack channel or formal ticketing (e.g., onboarding coordination, specific architectural discussions), you can reach out via email:

- **Email:** `idp-team@yourcompany.com` (Note: Replace with your actual team email address).