Phase 1 Implementation Plan: Foundational Infrastructure for the AI-Driven Longevity Coach

This document provides the definitive implementation plan for Phase 1 of the AI-Driven Longevity Coach project. It serves as a strategic and technical blueprint for establishing the core infrastructure on Google Cloud Platform (GCP) using Terraform. Adherence to this plan will ensure the creation of a secure, scalable, and automated foundation, perfectly aligned with the project's long-term objectives of dynamic AI content generation and robust service delivery.[1] Every recommendation herein is grounded in Google Cloud's best practices, emphasizing security, operational excellence, and future-readiness.

Section 1: GCP Environment and IAM Architecture

This section details the initial setup of the GCP environment, including the project itself and the critical Identity and Access Management (IAM) framework that will govern all resources. A robust and secure IAM strategy is the bedrock of the entire platform.

1.1. Project Provisioning and Governance

The foundational step is the creation of a dedicated Google Cloud project. This ensures that all resources, billing, permissions, and APIs for the Longevity Coach platform are logically isolated from other initiatives. This isolation is a fundamental security and management best practice, as it prevents accidental cross-environment interference and simplifies cost tracking, security auditing, and policy enforcement.[2]

Instruction

A new GCP project will be provisioned programmatically using the google_project Terraform resource. Manual creation via the console is to be avoided to ensure the entire environment is defined as code from its inception.

Configuration Details

- **Project ID:** A globally unique, descriptive, and immutable identifier must be selected (e.g., longevity-coach-prod). The choice of Project ID is permanent and cannot be altered after creation, so it must be chosen with care to reflect the project's purpose for its entire lifecycle.[3] The Project ID must be between 6 and 30 characters and can only contain lowercase letters, numbers, and hyphens.[3]
- **Billing Account:** The project must be linked to an active GCP Billing Account upon creation. This is a prerequisite for enabling any billable APIs and provisioning resources like GKE clusters or Artifact Registry repositories.[4] This will be configured using the billing_account argument within the google_project resource.
- **Organization/Folder:** To ensure proper governance and inheritance of organization-wide security policies, the project will be created within the designated parent organization or folder resource using the org_id or folder_id argument.[3]

1.2. Strategic API Enablement via Terraform

For any GCP service to be utilized within a project, its corresponding API must first be enabled. To maintain a declarative and version-controlled infrastructure, all required APIs will be enabled explicitly through Terraform. This practice codifies the project's dependencies on specific GCP services and prevents both environment drift caused by manual "click-ops" and runtime errors that occur when Terraform attempts to create a resource for a service that is not yet activated.[7]

Instruction

The google_project_service Terraform resource will be used to enable each necessary API. A distinct resource block will be defined for every API, ensuring the project's service

dependencies are explicit and auditable.[9]

The table below serves as the definitive manifest of APIs required for Phase 1. This list provides absolute clarity for the Infrastructure as Code (IaC) implementation and acts as an audit-friendly record of active services, preventing "API sprawl" by ensuring only necessary services are enabled.

API Service Identifier	Friendly Name	Purpose in Project
iam.googleapis.com	Identity and Access Management (IAM) API	To programmatically manage service accounts, roles, and IAM policy bindings.
cloudresourcemanager.googleapis.com	Cloud Resource Manager API	To allow Terraform to manage the GCP project itself and its metadata.
serviceusage.googleapis.com	Service Usage API	A prerequisite for allowing Terraform to enable other APIs declaratively.
container.googleapis.com	Kubernetes Engine API	To create, configure, and manage the GKE cluster and its node pools.[10]
artifactregistry.googleapis.com	Artifact Registry API	To create and manage the private Docker repository for storing application container images.[10]
iamcredentials.googleapis.com	IAM Service Account Credentials API	Required for Workload Identity Federation to function, allowing trusted external identities to impersonate GCP service accounts.

1.3. A Least-Privilege IAM Framework

The security of the platform begins with a meticulously designed IAM strategy founded on the principle of least privilege. This principle dictates that any identity—whether a user, a service, or an automated process—should be granted only the minimum permissions necessary to perform its intended function.[12] This approach drastically minimizes the potential impact of a compromised credential. Instead of using broad, permissive roles, the implementation will utilize dedicated, single-purpose service accounts with granular role bindings.

1.3.1. Service Account Strategy and Role Definitions

Two distinct service accounts (SAs) will be created for Phase 1, each with a clearly defined role and a minimal set of permissions.

- **Terraform Service Account (sa-terraform):** This service account will be the designated identity for all infrastructure modifications. It will be used exclusively by the CI/CD pipeline (via Workload Identity Federation) to execute terraform apply. Its permissions will be scoped to allow the creation and management of all resources defined in the Terraform configuration, including other service accounts.
- **GKE Node Service Account (sa-gke-nodes):** A custom, non-default service account will be created to serve as the identity for the GKE worker nodes. This is a critical security hardening measure that replaces the overly permissive Compute Engine default service account.[12] By assigning a custom SA with minimal permissions, the potential for a compromised node to escalate privileges and access or modify other GCP resources is significantly reduced.

The following table is the cornerstone of the security strategy, providing a transparent and auditable specification for every permission granted. It forces a deliberate decision for each role binding, ensuring the principle of least privilege is implemented correctly.

Service Account Name	Purpose	Assigned IAM Role(s)	Justification	
sa-terraform	Identity for the CI/CD pipeline to execute Terraform	roles/resourcemanager.projectIamAdmin roles/container	Grants permissions to manage all project	serviceAccountUser allows this SA to act as other SAs if

	commands.	.admin roles/artifactregistry.admin roles/iam.serviceAccountUser roles/storage.admin	resources defined in Terraform. projectIamAdmin is required to manage IAM bindings for other SAs. container.admin and artifactregistry.admin are needed for GKE and Artifact Registry management.[14]	needed. storage.admin is required to manage the Terraform state bucket.
sa-gke-nodes	Identity for GKE worker nodes to interact with other GCP services.	roles/monitoring.viewer roles/logging.logWriter roles/artifactregistry.reader	Provides the absolute minimum permissions required for nodes to function: read monitoring data, write logs to Cloud Logging, and pull container images from Artifact Registry.[12] Critically, this SA has no permissions to create, modify, or delete any resources.	

1.3.2. Securing CI/CD with Workload Identity Federation

The authentication mechanism between the GitHub Actions CI/CD pipeline and GCP is a pivotal security decision. To achieve the highest level of security and align with modern best practices, this project will exclusively use **Workload Identity Federation (WIF)**. This approach eliminates the need to generate, manage, and store long-lived, static service account keys, which are a primary target for attackers and a significant operational burden.[16]

By using WIF, authentication is based on short-lived, automatically-generated OIDC tokens, and trust is established through verifiable claims about the workflow's context. This is not merely an authentication choice; it is a foundational security architecture decision that removes the largest single point of failure in most CI/CD systems.[18] Access control becomes entirely governed by auditable IAM policies and trust configurations that can be instantly modified or revoked. This allows for hyper-granular control, such as granting deployment permissions only to workflows running on the

main branch, thereby creating a robust security gate for production changes.[20] This architecture elevates the project's security posture from standard to exemplary from day one, directly fulfilling the "best practices for security" objective outlined in the project plan.[1]

Instruction

The following components will be configured in GCP using Terraform to establish the WIF trust relationship with the project's GitHub repository.

1. **Workload Identity Pool:** A google_iam_workload_identity_pool resource will be created. This pool acts as a container for external identity providers and serves as the trust anchor for the federation.
2. **Workload Identity Provider:** A google_iam_workload_identity_pool_provider resource will be configured within the pool. This provider will be specifically configured for GitHub, with its OIDC issuer URL set to https://token.actions.githubusercontent.com.[17]
3. **Attribute Mapping and Conditions:** The provider configuration will include precise attribute conditions to strictly limit which GitHub workflows can authenticate. This is the core of the security model. The conditions will grant trust *only* to workflows originating from the specific project repository and the main branch.[21] An example condition would be:
   assertion.repository == 'your-org/longevity-coach' && assertion.ref == 'refs/heads/main'.
   This ensures that a workflow running on a feature branch or in a forked repository cannot assume the GCP identity.

4. **IAM Policy Binding:** Finally, a google_iam_policy_binding will be created to grant the federated identity the roles/iam.workloadIdentityUser role on the sa-terraform service account. This binding allows the authenticated GitHub Action workflow—and only that specific workflow—to impersonate the sa-terraform SA and temporarily inherit its permissions to execute Terraform commands.[21]

Section 2: Infrastructure as Code (IaC) Repository and Terraform Strategy

This section defines the structure and best practices for the Terraform codebase itself, ensuring it is maintainable, scalable, and suitable for team collaboration.

2.1. Structuring the Monorepo for Code and Configuration

A single GitHub repository (a "monorepo") will be initialized to house all project artifacts. This approach simplifies dependency management and ensures that the application code, its deployment configuration (Kubernetes manifests), and the infrastructure it runs on are versioned together. This creates a single, unambiguous source of truth for the entire state of the application and its environment.[22]

Proposed Directory Structure

The repository will be organized with the following directory structure to maintain a clean separation of concerns:

```
/longevity-coach
├──.github/workflows/      # GitHub Actions CI/CD workflow definitions
│   └── cicd.yaml
├── app/                   # Source code for the "Hello World" FastAPI application
```

```
│   ├── main.py
│   └── requirements.txt
├── infra/                  # All Terraform Infrastructure as Code
│   ├── environments/
│   │   └── prod/           # Configuration for the production environment
│   │       ├── main.tf
│   │       ├── variables.tf
│   │       ├── backend.tf
│   │       ├── gcp_project.tf
│   │       ├── network.tf
│   │       ├── gke.tf
│   │       └── apis.tf
├── k8s/                    # Kubernetes manifests for application deployment
│   ├── deployment.yaml
│   ├── service.yaml
│   └── ingress.yaml
└── Dockerfile              # Dockerfile for building the application container
```

2.2. Configuring a Secure Terraform Backend on GCS

Terraform requires a "state file" to map the resources in the code to the real-world resources in GCP. Storing this state file locally is unsuitable for team collaboration or automated pipelines. Therefore, a remote backend is mandatory.[2]

Instruction

Terraform's state will be stored securely and remotely in a dedicated Google Cloud Storage (GCS) bucket.

Configuration Details

- **GCS Bucket:** A new GCS bucket will be created solely for the purpose of storing the Terraform state file (terraform.tfstate).
- **Object Versioning:** Versioning will be enabled on this bucket. This provides a crucial

safety net, creating a history of all state file versions and allowing for recovery in case of accidental deletion or state corruption.[2]

- **State Locking:** The GCS backend natively supports state locking. This mechanism prevents multiple users or pipeline runs from executing terraform apply simultaneously, which could otherwise lead to a corrupted state file and infrastructure drift.[24]

The IaC Bootstrap Procedure

A classic "chicken-and-egg" problem exists when managing infrastructure as code: the resource used for the Terraform backend (the GCS bucket) must exist before Terraform can use it, but the goal is to create all resources with Terraform. To resolve this cleanly, a two-step bootstrap process will be executed once at the very beginning of the project.

1. **Step 1 (Manual Bootstrap):** A designated operator, authenticated locally via gcloud auth application-default login [25], will execute a minimal, one-time Terraform configuration. This configuration will be separate from the main project repository and its sole purpose is to create two resources: the GCS bucket for the remote state and the sa-terraform service account.
2. **Step 2 (Automated IaC):** Once the bootstrap is complete, the main Terraform configuration within the monorepo will be configured to use this pre-existing GCS bucket for its backend. All subsequent terraform plan and terraform apply commands will be executed by the CI/CD pipeline, which will authenticate as the sa-terraform service account via Workload Identity Federation.

This documented procedure establishes a clean and secure separation between the initial bootstrap environment and the fully automated, version-controlled infrastructure that will be managed thereafter.

2.3. Modular Terraform Code Organization

To ensure the Terraform codebase remains readable, maintainable, and scalable as the project grows, it will adhere to established best practices for structure and style.

Best Practices

- **Logical File Organization:** The Terraform code within the infra/environments/prod/ directory will be broken down into logical files based on the resources they manage (e.g., network.tf for VPC resources, gke.tf for the cluster, apis.tf for service enablement). This avoids monolithic .tf files and makes the code easier to navigate.[26]
- **Variable Definitions:** A clear distinction will be made between different types of variables. Environment-specific values that should not be committed to the repository (e.g., project_id, billing_account_id) will be passed in as CI/CD variables. Variables that define the configuration but may have sensible defaults (e.g., GKE node machine types, regions) will be defined in variables.tf with default values.[26]
- **Version Pinning:** To guarantee predictable and repeatable builds, the Terraform CLI version and all provider versions (e.g., hashicorp/google) will be explicitly pinned using required_version and required_providers blocks in the configuration. This prevents unexpected breaking changes from automatic updates of underlying tools and providers, which could otherwise cause pipeline failures.[2]

Section 3: Core Infrastructure Provisioning

This section details the Terraform configurations for provisioning the primary GCP resources that form the backbone of the platform: the virtual network, the container artifact repository, and the Kubernetes cluster.

3.1. Networking Foundation: A Secure VPC for GKE

A custom Virtual Private Cloud (VPC) network will be created to provide a secure and isolated networking environment for the GKE cluster. Using a custom VPC, rather than the default network, provides granular control over IP ranges, firewall rules, and network topology, which is a foundational security best practice.

Configuration Details

- **VPC:** A new VPC will be provisioned using the google_compute_network resource. Auto-creation of subnets will be disabled to allow for explicit subnet definition.
- **Subnet:** A primary subnet will be created in the target region using the

google_compute_subnetwork resource. This subnet will define the primary IP address range for the GKE nodes.
- **Secondary IP Ranges:** The GKE cluster will be configured as a VPC-native cluster, which requires dedicated IP ranges for Pods and Services. Two secondary IP ranges will be defined on the subnet for this purpose: one for Pod IP addresses and one for Kubernetes Service IP addresses. This allows Pod IPs to be natively routable within the VPC.
- **Firewall Rules:** GKE automatically creates the necessary firewall rules for cluster operation. However, default VPC rules, such as default-allow-internal, will be reviewed and potentially removed or restricted to enforce a more stringent, least-privilege network security posture.

3.2. Establishing the Artifact Registry Repository

A private, secure repository is required to store the Docker container images that the CI/CD pipeline will build. Artifact Registry is Google Cloud's next-generation service for managing container images and language packages, and it is the recommended choice over the legacy Container Registry (GCR).

Configuration Details

- **Resource:** The repository will be created using the google_artifact_registry_repository Terraform resource.
- **Format:** The repository format will be explicitly set to DOCKER.
- **Location:** To optimize for performance and cost, the repository will be created in the same GCP region as the GKE cluster. This minimizes network latency during image pulls (kubectl deploying a new pod) and avoids inter-regional data transfer costs.

3.3. Google Kubernetes Engine (GKE) Cluster Design and Configuration

The GKE cluster is the heart of the application's runtime environment. The configuration specified here prioritizes security, operational simplicity, and scalability, aligning with the project's long-term objectives.

3.3.1. Operational Mode: GKE Autopilot

The cluster will be provisioned in **Autopilot** mode. This is a strategic decision that directly supports the project's primary goal of focusing on AI agent development rather than complex Kubernetes infrastructure management.[1] Autopilot abstracts away the underlying node infrastructure, with Google managing node provisioning, scaling, upgrades, and security patching. This operational model provides a secure-by-default configuration, automatically including critical security features like Shielded GKE Nodes and a container-optimized OS with a hardened kernel, which would otherwise require extensive manual configuration and ongoing maintenance in a Standard GKE cluster.[13] By trading granular node-level control for significantly reduced operational overhead and an enhanced security posture, the development team can maximize its velocity and focus on building the core application and AI features that deliver business value.

3.3.2. Hardened Cluster Configuration

Even within the managed Autopilot model, several key configurations will be specified to further harden the cluster's security posture.

- **Networking:** The cluster will be configured as a **Private Cluster**. This ensures that the underlying nodes are provisioned without public IP addresses, making them inaccessible from the public internet and drastically reducing the potential attack surface. Access to the Kubernetes control plane (API server) will be restricted using **Authorized Networks**. This feature acts as a firewall, ensuring that kubectl commands and API calls can only originate from a pre-approved list of CIDR blocks, such as corporate VPNs or the specific IP ranges used by the CI/CD runners.[13]
- **Release Channel:** To ensure the cluster benefits from a balance of new features and proven stability, it will be enrolled in the REGULAR release channel. This delegates the management of Kubernetes versioning and security patching to Google, guaranteeing that the cluster is kept up-to-date automatically without manual intervention.
- **Service Account:** The cluster will be explicitly configured to use the custom, least-privilege sa-gke-nodes service account defined in Section 1.3.1, adhering to the principle of least privilege for node identity.

3.3.3. Enabling Workload Identity for In-Cluster Services

The GKE cluster will be created with **Workload Identity enabled**. This is a foundational step that prepares the cluster for the secure execution of the AI agents in later project phases. Workload Identity is the recommended method for Kubernetes workloads to securely access other Google Cloud services.[27] It works by linking a Kubernetes Service Account (KSA) within the cluster to a Google Service Account (GSA) in IAM. A pod running with the designated KSA can then automatically authenticate to GCP APIs as the corresponding GSA, acquiring short-lived credentials without ever needing to handle or mount static SA key files as Kubernetes secrets. This enables a secure, keyless, and auditable pattern for the AI agents (running in pods) to interact with services like Vertex AI and Firestore.

Section 4: "Hello World" Application Deployment Blueprint

To validate the end-to-end functionality of the infrastructure and CI/CD pipeline, a simple "Hello World" application will be created and deployed. This section specifies the application code, its containerization, and its Kubernetes deployment manifests.

4.1. FastAPI Application and Dockerfile Specification

- **Application:** A minimal web application will be created using the FastAPI framework. The file app/main.py will define a single API endpoint at the root path (/) that returns a simple JSON response, such as {"message": "Hello World"}.[29]
- **Dependencies:** The required Python packages, fastapi and uvicorn, will be listed in the app/requirements.txt file.
- **Dockerfile:** A multi-stage Dockerfile will be defined to build an optimized and secure container image.
    - **Stage 1 (Builder):** This stage will use a full Python base image (e.g., python:3.9) to install the application dependencies from requirements.txt into a dedicated directory.
    - **Stage 2 (Runtime):** This stage will start from a lightweight, slim base image (e.g., python:3.9-slim). It will copy the pre-installed dependencies from the builder stage and the application source code from app/. This multi-stage approach ensures that build-time tools and dependencies are not included in the final image, significantly reducing its size and attack surface.
    - **CMD Instruction:** The container's startup command will use the exec form: CMD

["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]. Using the exec form (with square brackets) is critical because it ensures the uvicorn process runs as PID 1 within the container, allowing it to properly receive and handle signals like SIGTERM for graceful shutdown, which is essential for zero-downtime rolling updates in Kubernetes.[30]

4.2. Kubernetes Manifests: Deployment, Service, and Ingress

Three standard Kubernetes manifest files will be created in the k8s/ directory. These YAML files declaratively define how the application is run, networked, and exposed to the internet.

- **deployment.yaml:** This manifest will define a Kubernetes Deployment resource. It will instruct Kubernetes to maintain a desired number of replica Pods running the application container. A key field, spec.template.spec.containers.image, will serve as a placeholder that the CI/CD pipeline will dynamically update with the specific image tag (the Git commit SHA) of the newly built image.[31]
- **service.yaml:** This manifest will define a Service of type ClusterIP. A ClusterIP service provides a stable, internal IP address and a DNS name for the set of pods managed by the Deployment. This allows other components within the cluster to reliably connect to the application without needing to know the ephemeral IP addresses of individual pods. The service will map its port (e.g., port 80) to the container's targetPort (8080).[31]
- **ingress.yaml:** This manifest will define an Ingress resource. The GKE Ingress controller will automatically detect this resource and provision a Google Cloud External HTTP(S) Load Balancer. The Ingress will contain rules to route external traffic arriving at the load balancer's public IP address to the internal ClusterIP Service, thereby exposing the application to the public internet.[32]

4.3. Initial CI/CD Workflow with GitHub Actions

A GitHub Actions workflow, defined in .github/workflows/cicd.yaml, will automate the entire build, push, and deploy process. This pipeline is the engine of continuous delivery for the project.

Workflow Steps

1. **Trigger:** The workflow will be configured to trigger automatically on every push event to the main branch.
2. **Checkout Code:** The first step in any job will be to use the actions/checkout action to check out the repository's source code onto the GitHub Actions runner.
3. **Authenticate to GCP:** The workflow will use the google-github-actions/auth action. This action will be configured to use Workload Identity Federation, leveraging the trust relationship established in Section 1.3.2 to securely obtain short-lived GCP credentials without the need for static keys stored as secrets.[34]
4. **Setup Tools:** The workflow will install and configure the necessary command-line tools, primarily gcloud and kubectl.
5. **Build and Push Docker Image:**
    - The runner will first authenticate its Docker client to the project's Artifact Registry using the command gcloud auth configure-docker.
    - It will then execute a docker build command, using the Dockerfile in the repository root.
    - The built image will be tagged with the unique Git commit SHA (${{ github.sha }}) to ensure every version is uniquely identifiable and traceable back to a specific code change.
    - Finally, the tagged image will be pushed to the Artifact Registry repository created in Section 3.2.[35]
6. **Deploy to GKE:**
    - The workflow will retrieve the credentials for the GKE cluster using gcloud container clusters get-credentials.
    - It will then dynamically update the k8s/deployment.yaml manifest file, replacing the placeholder image tag with the current Git commit SHA. This can be accomplished using tools like kustomize or a simple sed command.
    - The final step is to execute kubectl apply -f k8s/. This command sends the updated manifests to the Kubernetes API server, which will recognize the change in the Deployment's image tag and initiate a rolling update, gracefully replacing the old application pods with new ones running the updated container image.[19]

Section 5: Validation and Path to Phase 2

This final section outlines the objective criteria for the successful completion of Phase 1 and articulates how the established foundation directly enables the more complex features planned for subsequent project phases.

5.1. End-to-End Validation Procedures

The successful completion of Phase 1 is not measured by the individual creation of resources, but by the seamless, end-to-end functioning of the automated pipeline.

Definition of Done

Phase 1 is considered complete when a code change pushed to the main branch of the GitHub repository automatically and successfully triggers the entire CI/CD workflow, resulting in the updated "Hello World" application being served live from the GKE cluster without any manual intervention.

Verification Steps

1. **CI/CD Pipeline Success:** Confirm that the GitHub Actions workflow triggered by the push completes all its jobs successfully, with green checkmarks for each step.
2. **Artifact Verification:** Navigate to the Artifact Registry repository in the GCP console and verify that a new container image has been pushed, tagged with the Git commit SHA of the latest push.
3. **Deployment Verification:** Using kubectl get deployment and kubectl describe deployment, confirm that the application's Deployment in GKE has successfully rolled out and that its pods are in a Running state and are using the new image tag.
4. **Application Accessibility:** Retrieve the public IP address of the Ingress-managed Load Balancer (kubectl get ingress). Access this IP address in a web browser or via curl and verify that the expected "Hello World" JSON response is returned from the live application.

5.2. How This Foundation Enables Future Phases

The infrastructure established in Phase 1 is not merely a "Hello World" setup; it is a production-grade foundation specifically designed to support the future needs of the

AI-Driven Longevity Coach platform.

- **Scalability for AI Workloads:** The GKE Autopilot foundation is inherently scalable. As the background AI agents are developed in Phases 3 and 4, their computational demands (e.g., for processing research papers or running LLM inferences) can be met by Autopilot automatically scaling the underlying compute resources up or down, without requiring manual node pool management.
- **Secure API Integration:** The enablement of Workload Identity on the cluster is a critical prerequisite for Phase 3. When the AI agents are deployed as pods, they will be able to securely authenticate to Vertex AI, Firestore, and other GCP APIs using this keyless mechanism. This provides a secure and manageable way for the application's core logic to interact with its cloud environment.
- **Velocity and Iteration:** The fully automated CI/CD pipeline is the engine for the project's agility, directly fulfilling the "Continuous Delivery with AI Assistance" objective.[1] As the website frontend is built (Phase 2), new AI agents are developed (Phase 3), and the Q&A coach is implemented (Phase 5), this pipeline will ensure that every improvement can be integrated, tested, and deployed to users rapidly and reliably. This robust automation closes the loop on the DevOps lifecycle and sets the stage for continuous innovation.

Works cited

1. Project Plan_ AI-Driven Longevity Coach Website on GCP.pdf
2. Terraform Best Practices on Google Cloud: A Practical Guide | by Cuong Truong | Medium, accessed September 19, 2025, https://medium.com/@truonghongcuong68/terraform-best-practices-on-google-cloud-a-practical-guide-057f96b19489
3. Creating and managing projects | Resource Manager Documentation - Google Cloud, accessed September 19, 2025, https://cloud.google.com/resource-manager/docs/creating-managing-projects
4. Create a Google Cloud project | Google Workspace - Google for Developers, accessed September 19, 2025, https://developers.google.com/workspace/guides/create-project
5. APIs and billing - API Console Help - Google Help, accessed September 19, 2025, https://support.google.com/googleapi/answer/6158867?hl=en
6. Enabling Google Cloud Billing | Openbridge Help Center, accessed September 19, 2025, https://docs.openbridge.com/en/articles/8179933-enabling-google-cloud-billing
7. Understanding Google Cloud APIs and Terraform | Infrastructure Manager, accessed September 19, 2025, https://cloud.google.com/docs/terraform/understanding-apis-and-terraform
8. Can I automatically enable APIs when using GCP cloud with terraform? - Stack Overflow, accessed September 19, 2025, https://stackoverflow.com/questions/59055395/can-i-automatically-enable-apis-when-using-gcp-cloud-with-terraform
9. google_project_service | Resources | hashicorp/google - Terraform Registry,

accessed September 19, 2025, https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/google_project_service
10. GCP List of API Services - GitHub Gist, accessed September 19, 2025, https://gist.github.com/coryodaniel/13eaee16a87a7fdca5e738123216862a
11. Artifact Registry API - Google Cloud, accessed September 19, 2025, https://cloud.google.com/artifact-registry/docs/reference/rest
12. Harden your cluster's security | GKE Documentation - Google Cloud, accessed September 19, 2025, https://cloud.google.com/kubernetes-engine/docs/how-to/hardening-your-cluster
13. Google Kubernetes (Container) Engine Cluster - Examples and best practices | Shisho Dojo, accessed September 19, 2025, https://shisho.dev/dojo/providers/google/Kubernetes_Container_Engine/google-container-cluster/
14. Provisioning Google Cloud Infrastructure with Terraform (GKE Cluster Example) - Terrateam, accessed September 19, 2025, https://terrateam.io/blog/how-to-build-gke-cluster-with-terraform
15. Access control with IAM | Artifact Registry documentation - Google Cloud, accessed September 19, 2025, https://cloud.google.com/artifact-registry/docs/access-control
16. How to use Github Actions with Google's Workload Identity Federation - YouTube, accessed September 19, 2025, https://www.youtube.com/watch?v=ZgVhU5qvK1M
17. Setting Up Workload Identity Federation Between GitHub Actions and Google Cloud Platform - Firefly, accessed September 19, 2025, https://www.firefly.ai/academy/setting-up-workload-identity-federation-between-github-actions-and-google-cloud-platform
18. Deploying to Google Kubernetes Engine - GitHub Enterprise Cloud Docs, accessed September 19, 2025, https://docs.github.com/zh/enterprise-cloud@latest/actions/how-tos/use-cases-and-examples/deploying/deploying-to-google-kubernetes-engine
19. Deploying to Google Kubernetes Engine - GitHub Docs, accessed September 19, 2025, https://docs.github.com/actions/deployment/deploying-to-google-kubernetes-engine
20. TERRAFORM AUTOMATION with GitHub and GCP Workload Identity Federation - YouTube, accessed September 19, 2025, https://www.youtube.com/watch?v=DMwI9WcSAL8
21. Configure Workload Identity Federation with deployment pipelines | IAM Documentation, accessed September 19, 2025, https://cloud.google.com/iam/docs/workload-identity-federation-with-deployment-pipelines
22. How to Deploy Kubernetes Resources with Terraform - Spacelift, accessed September 19, 2025, https://spacelift.io/blog/terraform-kubernetes-deployment

23. Adding GitHub Repo to Terraform - Reddit, accessed September 19, 2025, https://www.reddit.com/r/Terraform/comments/t30211/adding_github_repo_to_terraform/
24. 20 Terraform Best Practices to Improve your TF workflow - Spacelift, accessed September 19, 2025, https://spacelift.io/blog/terraform-best-practices
25. Best practices for Terraform operations | Google Cloud, accessed September 19, 2025, https://cloud.google.com/docs/terraform/best-practices/operations
26. Best practices for general style and structure | Terraform - Google Cloud, accessed September 19, 2025, https://cloud.google.com/docs/terraform/best-practices/general-style-structure
27. About cluster configuration choices | Google Kubernetes Engine (GKE), accessed September 19, 2025, https://cloud.google.com/kubernetes-engine/docs/concepts/configuration-overview
28. Install Harness Delegate on Google Kubernetes Engine (GKE) with Workload Identity, accessed September 19, 2025, https://developer.harness.io/docs/platform/delegates/install-delegates/gke-workload-identity/
29. First Steps - FastAPI, accessed September 19, 2025, https://fastapi.tiangolo.com/tutorial/first-steps/
30. FastAPI in Containers - Docker, accessed September 19, 2025, https://fastapi.tiangolo.com/deployment/docker/
31. Kubernetes Ingress. Project: Deploying a Web Application... | by Nidhi Ashtikar - Medium, accessed September 19, 2025, https://nidhiashtikar.medium.com/kubernetes-ingress-d71127920357
32. Kubernetes Ingress with NGINX Ingress Controller Example - Spacelift, accessed September 19, 2025, https://spacelift.io/blog/kubernetes-ingress
33. Kubernetes Ingress: A Practical Guide - Solo.io, accessed September 19, 2025, https://www.solo.io/topics/api-gateway/kubernetes-ingress
34. How to push tagged Docker releases to Google Artifact Registry with a GitHub Action, accessed September 19, 2025, https://gist.github.com/palewire/12c4b2b974ef735d22da7493cf7f4d37
35. Building and Pushing to Artifact Registry with Github Actions | by Sarah Kapelner - Medium, accessed September 19, 2025, https://medium.com/@sbkapelner/building-and-pushing-to-artifact-registry-with-github-actions-7027b3e443c1
36. Using GitHub Actions with Google Cloud Deploy, accessed September 19, 2025, https://cloud.google.com/blog/products/devops-sre/using-github-actions-with-google-cloud-deploy