# **We The Gamers: A Definitive Kubernetes Architectural Blueprint**

## **1. Cloud Provider Analysis & Recommendation**

### **1.1 Introduction**

The foundational decision for the "We The Gamers" platform is the choice of a cloud service provider (CSP). The platform's business model, centered on a low-cost $0.99 membership, dictates that every technical decision must be scrutinized through the lens of extreme cost optimization.1 This analysis will therefore not focus on the breadth of services offered by Amazon Web Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure, but rather on a narrow, critical set of metrics that will directly and disproportionately impact the platform's operational expenditure. These key factors are: managed Kubernetes control plane fees, the mechanics and pricing of spot instances, the price-performance of ARM-based compute, and the cost of egress network traffic.

### **1.2 Managed Kubernetes Control Plane Cost Analysis**

The control plane is the brain of a Kubernetes cluster. All three major CSPs offer a managed service: Amazon Elastic Kubernetes Service (EKS), Google Kubernetes Engine (GKE), and Azure Kubernetes Service (AKS).

* **AWS EKS:** Charges a flat rate of $0.10 per hour for each cluster's control plane, which translates to approximately $72 per month.2 This fee is constant regardless of cluster size and does not include a free tier.2
* **Azure AKS & Google GKE:** Both providers offer a free tier for the control plane. For AKS, the control plane is free by default but does not come with a financially-backed Service Level Agreement (SLA).5 GKE's free tier covers one zonal cluster per billing account.5 To achieve a production-grade SLA comparable to EKS, both AKS and GKE introduce a "Standard" tier, which also costs $0.10 per hour, effectively matching the price of EKS.7

For a startup operating on a lean budget, the free control plane offered by GKE and AKS presents an immediate cost saving of ~$72 per month. While this is an attractive tactical advantage during the initial development and bootstrapping phase, it is not a strategic one. The "We The Gamers" platform will offer a paid membership tier, which carries an implicit expectation of reliability and uptime. Operating a production service on a non-SLA-backed control plane introduces significant operational risk. Therefore, the architecture must assume an eventual migration to the SLA-backed tier, at which point the control plane costs across all three providers become identical. This neutralizes the initial cost advantage and shifts the focus of this analysis to the more impactful, long-term cost drivers: compute and data transfer.

### **1.3 Spot Instance and Preemptible VM Analysis**

The business plan mandates a free tier for lightweight, non-persistent game servers.1 The financial viability of this offering depends entirely on running these workloads on the cheapest possible compute. Spot Instances (AWS) and Preemptible/Spot VMs (GCP/Azure) offer discounts of up to 90% over on-demand pricing by utilizing spare cloud capacity.9 However, the mechanics of how these instances are reclaimed by the provider are critically different.

* **Pricing Volatility:** AWS Spot prices fluctuate dynamically based on supply and demand, with an average of 197 distinct price changes per month for a given instance type.9 In contrast, GCP and Azure prices are more stable, changing less than once a month on average.9
* **Interruption Notice:** AWS provides a **two-minute warning** before a Spot Instance is reclaimed.10 GCP and Azure provide only a  
  **30-second warning**.10

For the "We The Gamers" workload, the interruption notice period is a far more critical feature than price stability. The free-tier game servers, while ephemeral, must provide a positive user experience to serve as an effective funnel to the paid membership. A sudden server termination with only 30 seconds of notice is a jarring experience that risks corrupting game state and frustrating users. Many game servers, such as Minecraft, can take a significant amount of time to perform a graceful shutdown and save world data.

The two-minute warning provided by AWS is a decisive technical advantage. It provides a sufficient window for the Agones game server controller to detect the impending shutdown, send a notification to connected players, execute a world-save command, and gracefully terminate the server process. This ability to manage the shutdown process transforms a potentially negative user event into a controlled, communicated one. This operational benefit, which directly translates to user satisfaction and retention, far outweighs the inconvenience of AWS's more volatile spot pricing model.

### **1.4 ARM-based Instance Price-Performance**

A core strategy for aggressive cost reduction is the adoption of ARM-based processors, which consistently deliver superior price-performance over traditional x86 CPUs for a wide range of cloud workloads.9

* **AWS Graviton:** AWS has invested heavily in its custom-designed Graviton ARM processors. The latest generation, Graviton3, offers up to 25% better compute performance than Graviton2 and is supported across a wide and mature family of instance types (general purpose, compute-optimized, memory-optimized).14
* **GCP Tau T2A & Azure Dpsv5:** Both GCP and Azure utilize Ampere Altra ARM processors.16
* **Performance Benchmarks:** For network-intensive workloads analogous to API gateways and game server backends, independent benchmarks show AWS's Graviton3-powered c7g instances delivering significantly higher performance (up to 2x the queries per second) and better cost-performance than the Ampere-based offerings from GCP and Azure.14

The "We The Gamers" platform is architected to run as many workloads as possible on ARM instances. This includes not only the stateless web applications and core platform services but also the game servers themselves, particularly for titles like Minecraft that have excellent ARM support. The sustained 20-40% price-performance improvement offered by ARM is not a minor optimization; it is a fundamental reduction in the platform's largest and most significant operational cost.13 AWS's clear leadership, maturity, and superior performance in the ARM compute space represent a powerful and continuous long-term cost advantage.

### **1.5 Egress Traffic Cost Analysis**

As a platform whose primary function is to serve real-time game server traffic to users across the public internet, egress network traffic will be a major component of the monthly cloud bill.18

* **Free Tier:** As of 2025, AWS provides the most generous free tier for data transfer out to the internet, at **100 GB per month**. This is significantly larger than the 5 GB per month offered by both Azure and GCP.19
* **Standard Rates:** Beyond the free tier, the initial per-gigabyte rates are broadly comparable across the providers, with AWS at ~$0.09/GB, Azure at ~$0.087/GB, and GCP at ~$0.12/GB.19

While the per-GB rates converge, the larger 100 GB free tier from AWS provides a longer financial runway during the critical startup phase when user numbers are growing but revenue is minimal. This buffer is a tangible benefit for a business model that is highly sensitive to data transfer costs.

### **1.6 Definitive Recommendation**

Based on a holistic analysis of the cost factors that will scale with the platform's usage, **Amazon Web Services (AWS) is the definitive recommendation.**

While the free control planes of GKE and AKS are initially appealing, they represent a small, fixed saving that is overshadowed by the substantial, compounding cost advantages offered by AWS in the areas that truly matter for this specific workload. The final decision is a strategic one, justified by the combined and sustained impact of three key technical differentiators.

The following table summarizes the decisive factors in this analysis.

| Feature | AWS (EKS) | GCP (GKE) | Azure (AKS) | Justification for "We The Gamers" |
| --- | --- | --- | --- | --- |
| **Control Plane Cost (SLA)** | ~$72/month | ~$72/month | ~$72/month | A production service requires an SLA, neutralizing the free-tier offerings of GCP/Azure and making this a non-differentiating factor for long-term planning.5 |
| **Spot Instance Model** | **Superior** (2-min warning) | Inferior (30-sec warning) | Inferior (30-sec warning) | The 2-minute interruption notice is a critical technical enabler for gracefully managing stateful free-tier game servers, directly improving user experience and retention.10 |
| **ARM Price-Performance** | **Superior** (Graviton) | Good (Ampere) | Good (Ampere) | AWS's mature Graviton ecosystem and superior performance provide a sustained 20-40% cost reduction on compute, the platform's single largest expense.13 |
| **Egress Free Tier** | **Superior** (100 GB/month) | Inferior (5 GB/month) | Inferior (5 GB/month) | The larger free tier provides a longer financial runway for a business model highly dependent on internet data transfer.19 |
| **Final Recommendation** | **Recommended** | Not Recommended | Not Recommended | The combined, long-term cost savings from a superior spot model, leading ARM price-performance, and a more generous egress free tier far outweigh the minor initial saving on the control plane offered by competitors. |

## **2. Cluster Provisioning & Automation**

### **2.1 Introduction**

A fully automated, repeatable, and version-controlled infrastructure is non-negotiable for a platform designed to be managed by a single engineer.1 We will adhere strictly to the principles of Infrastructure as Code (IaC) using HashiCorp Terraform to provision the entire AWS EKS ecosystem. This approach eliminates manual configuration, which is error-prone and unscalable, and ensures that the infrastructure can be recreated consistently from a single source of truth: the code in our Git repository.22 We will leverage the official and well-maintained

terraform-aws-modules/eks/aws module, which provides a production-grade foundation for creating the cluster and its associated resources.23

### **2.2 VPC and Networking Setup**

The foundation of the EKS cluster is a well-architected Virtual Private Cloud (VPC). For security and resource isolation, a new VPC will be created specifically for this platform, avoiding the use of the default AWS VPC.22

* **High Availability:** The VPC will be configured to span three Availability Zones (AZs). This multi-AZ design is fundamental to the high-availability strategy for both the Kubernetes control plane and the worker nodes.
* **Subnet Strategy:** Within each AZ, two types of subnets will be created:
  + **Public Subnets:** These subnets will have a route to an Internet Gateway. They will house the NAT Gateways required for outbound internet connectivity. No worker nodes will be placed here.
  + **Private Subnets:** These subnets will not have a direct route to the internet. All worker nodes will be launched in these private subnets to shield them from direct exposure to public traffic. Their outbound internet access (e.g., for pulling container images) will be routed through the NAT Gateways in the public subnets.
* **CIDR Planning:** The VPC's primary CIDR block will be chosen with foresight (e.g., /16) to provide a large IP address space. The AWS VPC CNI plugin, used by EKS, assigns a real VPC IP address to each pod. Insufficient IP space is a common scaling problem, and planning for a large address space from the outset prevents complex network re-architecting in the future.22

### **2.3 EKS Cluster Provisioning**

Using the Terraform EKS module, the control plane will be provisioned with the following production-grade configurations:

* **Endpoint Security:** The cluster will be configured with cluster\_endpoint\_private\_access = true and cluster\_endpoint\_public\_access = false. This is a critical security measure that removes the Kubernetes API server from the public internet. All administrative access (e.g., kubectl) must then be routed through a bastion host or VPN connection within the VPC, drastically reducing the attack surface.22
* **Control Plane Logging:** All available control plane log types (api, audit, authenticator, controllerManager, scheduler) will be enabled and configured to ship to Amazon CloudWatch Logs. The audit log, in particular, is essential for security analysis, providing a detailed chronological record of all actions performed against the API server.

### **2.4 Node Group Architecture: The Core of the Tiered Service Model**

The architecture's elegance lies in its direct mapping of the business model's service tiers to distinct, specialized infrastructure tiers. This is achieved through a carefully designed set of EKS Managed Node Groups, each with specific instance types, pricing models, and Kubernetes taints to enforce workload separation.1 Taints are applied to nodes to repel pods, while tolerations are applied to pods to allow them to be scheduled on nodes with matching taints.24 This mechanism is the cornerstone of how we will enforce our service tiers automatically.

This physical separation of workloads is not merely for organizational purposes; it is a fundamental control mechanism for both cost and quality of service. Free-tier users are programmatically siloed onto inexpensive, interruptible hardware, while paying members are guaranteed access to premium, stable resources. This is the technical implementation that makes the "gaming commune socialist spirit" of a free tier financially sustainable.

The following table details the configuration of each node group.

| Node Group Name | Purpose | Instance Types | Pricing Model | Taints |
| --- | --- | --- | --- | --- |
| core-services | Runs critical platform services (ArgoCD, Vault, Monitoring, Database Operator, Web Apps) | t4g.\* (ARM) | On-Demand / Reserved | CriticalAddonsOnly=true:NoSchedule |
| gameserver-free | Exclusively runs free-tier, ephemeral game servers on the cheapest possible compute. | t4g.\*, t3.\* (ARM & x86) | **Spot Instances** | gameserver-tier=free:NoSchedule |
| gameserver-member | Runs persistent, higher-resource game servers for paying members. | m7g.\*, c7g.\* (ARM) | On-Demand / Reserved | gameserver-tier=member:NoSchedule |

* **core-services Node Group:** This group forms the stable backbone of the platform. It runs on reliable On-Demand or, for further cost savings, Reserved t4g Graviton instances.13 The  
  CriticalAddonsOnly=true:NoSchedule taint ensures that only pods with a corresponding toleration—which we will apply to our core service manifests—can be scheduled here, preventing game server workloads from consuming critical platform resources.24
* **gameserver-free Node Group:** This is the workhorse for the free tier and is engineered for maximum cost savings. It will be configured as an Auto Scaling Group using a **mixed-instances policy**. This allows it to request Spot capacity from a prioritized list of instance types and families (e.g., t4g.medium, t3.medium, m6i.large), dramatically increasing the probability of acquiring spot capacity at the lowest price.10 The  
  gameserver-tier=free:NoSchedule taint acts as a gatekeeper, ensuring only pods explicitly tolerating this taint (i.e., our free-tier Agones game servers) can be scheduled on this volatile but extremely cheap hardware.25
* **gameserver-member Node Group:** This group provides the premium experience for paying members. It uses stable On-Demand or Reserved Instances from the memory-optimized (m7g) or compute-optimized (c7g) Graviton families to guarantee availability and performance.13 The  
  gameserver-tier=member:NoSchedule taint reserves these nodes exclusively for workloads that tolerate it, ensuring that paying members' servers are not competing for resources with the free tier or core services.

### **2.5 Automation Wrapper**

To simplify the management of the infrastructure for the single engineer, a simple Makefile will be created in the root of the Terraform project. This file will provide convenient targets that wrap the standard Terraform commands, such as make plan, make apply, and make destroy. This abstracts away the need to remember specific command-line flags and provides a simple, repeatable, one-command interface for managing the entire infrastructure lifecycle.26

## **3. Enterprise-Grade Security Architecture**

### **3.1 Introduction**

For a multi-tenant platform that will handle user data and execute potentially untrusted code in the form of game servers, security cannot be an afterthought; it must be woven into the fabric of the architecture from the very beginning. The following sections outline a defense-in-depth strategy founded on the principle of zero trust, where no component is trusted by default and all access must be explicitly granted, authenticated, and audited.

### **3.2 Centralized Secrets and Certificate Management with HashiCorp Vault**

Storing sensitive information like database passwords, API keys, or TLS certificates directly in Git or as unencrypted Kubernetes Secret objects is a significant security vulnerability. To mitigate this, a highly available HashiCorp Vault cluster will be deployed to serve as the single, secure source of truth for all secrets.

* **What it is:** Vault is a tool for securely accessing secrets. It provides a centralized interface to manage secrets, policies, and identities, while enforcing strict access control and recording a detailed audit log of all interactions.27
* **Why it's important:** It moves secrets out of developer workflows and application code, provides dynamic secret generation (e.g., temporary database credentials), encrypts data both in transit and at rest, and allows for fine-grained access policies.
* **Actionable Best Practices:**
  1. **HA Deployment:** A three-node Vault cluster will be deployed onto the core-services node group using the official Vault Helm chart. This ensures the secrets management layer itself is highly available.27
  2. **Kubernetes Authentication:** The Vault Kubernetes authentication method will be enabled and configured.28 This is a cornerstone of the integration. It allows a Kubernetes pod to authenticate with Vault by presenting its Service Account Token (SAT). Vault then validates this token with the Kubernetes API server and, if successful, issues a temporary Vault token with policies attached to a pre-configured role.
  3. **Secrets Engines:** The KVv2 (Key-Value Version 2) secrets engine will be used for storing static application secrets, providing versioning and undelete capabilities. The PKI (Public Key Infrastructure) engine will be configured to act as an internal Certificate Authority (CA), dynamically generating short-lived TLS certificates for securing inter-service communication (mTLS).
  4. **Vault Agent Injector:** The Vault Agent Injector is a critical component that decouples applications from Vault awareness.30 It runs as a Kubernetes Mutating Admission Webhook that intercepts pod creation events. If a pod is annotated correctly (e.g.,  
     vault.hashicorp.com/agent-inject: 'true'), the injector automatically adds two containers to the pod:
     + An **init container** that authenticates with Vault, retrieves the requested secrets, and writes them to a shared in-memory volume before the main application container starts.
     + A sidecar container that keeps the Vault token renewed and watches for changes to the secrets, updating the files on the shared volume dynamically.  
       This pattern means application developers do not need to write any custom code to interact with Vault; they simply read secrets from a file path (e.g., /vault/secrets/database.txt) as if they were mounted from a standard ConfigMap or Secret.31

### **3.3 Tiered Role-Based Access Control (RBAC)**

A default Kubernetes cluster is often overly permissive. A tiered RBAC strategy is essential to enforce the principle of least privilege, ensuring that users, groups, and service accounts have only the permissions they absolutely need to perform their functions.32 This is crucial for preventing privilege escalation and for safely delegating tasks to community administrators.

* **Example Roles and ClusterRoles:**
  + **superadmin (ClusterRole):** This is the highest-privilege role, granting \* access to all resources across all API groups in the entire cluster. It will be bound *only* to the IAM user of the lead engineer. This is the "break glass in case of emergency" role and is not used for daily operations.33
  + **server-admin (Role):** This is a namespaced Role that will be bound to the Service Account used by the Discord bot's backend API. It is the key to secure delegation. This role will be granted an extremely limited set of permissions, for example:
    - apiGroups: ["agones.dev"], resources: ["gameservers/status"], verbs: ["patch"] (to allow shutting down a server by patching its state).
    - apiGroups: ["agones.dev"], resources: ["gameserverallocations"], verbs: ["create"] (to request a new server from a fleet).
    - apiGroups: [""], resources: ["pods/log"], verbs: ["get", "list", "watch"] (to allow viewing logs for specific game server pods).  
      This role will have NO permissions to modify Deployments, Nodes, Secrets, or any other core cluster infrastructure.32
  + **cicd-system (Role):** This namespaced Role will be bound to the Service Account used by the ArgoCD Application Controller. It will have permissions to manage application lifecycle resources (Deployments, Services, Ingresses, etc.) but *only* within designated application namespaces (e.g., web-apps, discord-bot). It will be explicitly denied access to critical system namespaces like kube-system, vault, and monitoring.

This RBAC structure is what enables the business model of delegated administration. Community admins, interacting through the Discord bot, are non-technical and represent a potential vector for error or abuse. The server-admin role, combined with a secure API gateway for the bot, creates a secure operational chokepoint. The admins can only perform the actions exposed by the bot's API, and that API is backed by a Service Account whose permissions are surgically limited by RBAC. They cannot run kubectl exec, delete a critical deployment, or access Vault, because the technical controls at the Kubernetes API level make it impossible. This is how powerful operational capabilities are delegated safely to a less-trusted user group.

### **3.4 Network Policies**

By default, Kubernetes allows all pods to communicate with all other pods across all namespaces, which represents an unacceptably large attack surface. A "default-deny" network policy posture will be enforced using Calico as the Container Network Interface (CNI).

* **What it is:** Network Policies are Kubernetes resources that control the traffic flow at the IP address or port level (Layer 3/4). Calico is a powerful CNI that supports these policies and more advanced features.34
* **Why it's important:** A default-deny stance ensures that no traffic is allowed unless it is explicitly permitted. This prevents lateral movement by an attacker if a single pod is compromised.
* **Actionable Best Practices:**
  1. **Install Calico:** Calico will be installed as the CNI during the cluster's bootstrapping process.
  2. **Global Default-Deny:** A GlobalNetworkPolicy will be applied with a namespaceSelector that targets all application namespaces but explicitly excludes critical system namespaces like kube-system, calico-system, and vault. This policy will have no ingress or egress rules, effectively blocking all traffic by default.34
  3. **Allow Essential Traffic:** A second, high-priority GlobalNetworkPolicy will be created to allow essential cluster-wide traffic, most notably DNS queries on UDP/TCP port 53 to pods with the k8s-app: kube-dns label.34
  4. **Granular Application Policies:** For every application deployed, a specific NetworkPolicy will be crafted. For example, a policy for the frontend-api pods might look like this:
     + **Ingress:** Allow traffic on port 443 only from pods with the label app: ingress-controller.
     + Egress: Allow traffic only to pods with the label app: backend-api on port 8080.  
       All other ingress and egress traffic to and from the frontend-api pods would be blocked by the default-deny policy.

### **3.5 Pod Security Standards (PSS)**

Pod Security Standards are a built-in Kubernetes admission control mechanism that prevents pods from being created with insecure configurations. They define three profiles: privileged, baseline, and restricted.

* **What it is:** A cluster-level enforcement of security contexts for pods.
* **Why it's important:** The restricted profile provides the strongest hardening, preventing common container breakout and privilege escalation techniques by disallowing pods from running as root, using host networking, or accessing the host filesystem.
* **Actionable Best Practices:**
  1. **Enforce Restricted by Default:** The Kubernetes API server's admission control configuration will be set to enforce the restricted PSS profile cluster-wide by default. This means any new namespace created will automatically inherit this strict policy.
  2. **Labeled Exceptions:** For the few system namespaces that require slightly elevated permissions (e.g., kube-system, calico-system, monitoring), they will be explicitly labeled to allow the baseline profile. No workload on the cluster will be permitted to run with the privileged profile.

## **4. GitOps & CI/CD Workflow**

### **4.1 Introduction**

To achieve the high degree of automation required for a single-engineer operation, a pure GitOps workflow will be adopted. The Git repository will serve as the single, declarative source of truth for the entire desired state of the Kubernetes cluster. This methodology eliminates manual, imperative changes (kubectl apply -f...), which are error-prone, difficult to audit, and impossible to roll back cleanly. ArgoCD will be the engine that powers this workflow.

### **4.2 GitOps with ArgoCD**

* **What it is:** ArgoCD is a declarative, GitOps continuous delivery tool for Kubernetes. It runs as a controller in the cluster, continuously monitoring a specified Git repository. When it detects a difference between the state defined in the Git repository's manifests and the live state in the cluster, it automatically takes action to reconcile the cluster to match the desired state in Git.36
* **Why it's important:**
  + **Declarative:** You define *what* you want, not *how* to get there.
  + **Auditability:** Every change to the cluster is a Git commit, providing a complete, auditable history of who changed what, when, and why.
  + **Consistency & Reliability:** Eliminates configuration drift and ensures environments can be reproduced reliably.
  + **Easy Rollbacks:** A bad deployment can be reverted by simply executing git revert on the problematic commit.
* **Actionable Best Practices:**
  1. ArgoCD will be installed into its own argocd namespace on the core-services node group using its official Helm chart.36
  2. The ArgoCD web UI will be exposed via a secure Ingress for administrative visibility and manual sync/refresh operations.
  3. ArgoCD will be configured with read-only credentials to connect to the platform's central GitOps repository.

### **4.3 Git Repository Structure: The App of Apps Pattern**

A monolithic directory of YAML manifests quickly becomes unmanageable. To solve this, we will use the "App of Apps" pattern, a hierarchical approach where a single "root" ArgoCD Application resource points to other Application resources, which in turn point to the actual workload manifests.38 This creates a logical, scalable, and maintainable structure.

* **Repository Layout:**  
  gitops-repo/  
  ├── bootstrap/  
  │ └── root-app.yaml # The single root app pointing to directories below  
  └── applications/  
   ├── 01-platform/ # Core platform services (monitoring, security, etc.)  
   │ ├── monitoring.yaml # App manifest for kube-prometheus-stack  
   │ ├── security.yaml # App manifest for Vault, Calico policies  
   │ └── database.yaml # App manifest for the Postgres operator  
   ├── 02-workloads/ # Our custom applications  
   │ ├── web-apps.yaml # App manifest for all web frontends/APIs  
   │ └── discord-bot.yaml # App manifest for the RedBot application  
   └── 03-games/ # Game server configurations  
   └── games-appset.yaml # The ApplicationSet for all game fleets
* **Manifests Storage:** The actual Kubernetes manifests (Deployments, Services, etc.) for each application will be stored in separate subdirectories, managed with Kustomize to handle environment-specific configurations (e.g., dev vs. prod). The ArgoCD Application manifests in the structure above will point to these Kustomize directories.

### **4.4 CI/CD Pipeline**

The GitOps workflow cleanly separates the Continuous Integration (CI) process from the Continuous Delivery (CD) process. CI is responsible for building and testing artifacts, while CD (handled by ArgoCD) is responsible for deploying them.

* **Pipeline Stages (Using GitHub Actions):**
  1. **Commit:** A developer pushes code to an application's source code repository (e.g., the backend-api repository).
  2. **CI - Test & Analyze:** A GitHub Actions workflow is triggered. It runs unit tests, integration tests, code linting, and static security analysis (SAST).
  3. **CI - Build & Push Image:** Upon successful testing, the workflow builds a new container image using a multi-stage Dockerfile to keep the final image lean. The image is tagged with the Git commit SHA for traceability.
  4. **CI - Push to Registry:** The newly built and tagged image is pushed to a dedicated Amazon ECR (Elastic Container Registry) repository.
  5. **CD - Update Manifest Repository:** This is the hand-off from CI to CD. A final job in the GitHub Actions workflow checks out the separate gitops-repo. It uses a tool like Kustomize to programmatically update the Deployment manifest for the application, setting the image tag to the new commit SHA. It then commits and pushes this change to the gitops-repo.
  6. **GitOps - Sync & Deploy:** ArgoCD, which is constantly monitoring the gitops-repo, detects the new commit. It recognizes that the live Deployment in the cluster has an old image tag, while the desired state in Git has the new one. ArgoCD then applies the updated manifest to the cluster, which triggers a safe, rolling update of the application's pods.

### **4.5 Scalable Game Management with ApplicationSet**

As the platform grows to support more games and tiers, manually creating and managing an ArgoCD Application for each game's Agones Fleet becomes operationally burdensome.1 The ArgoCD ApplicationSet controller is the solution for this, enabling the templatization and automated generation of

Application resources.38

The Git Directory Generator within ApplicationSet is particularly powerful for this use case. We can structure a games/ directory within our manifest storage area where each subdirectory represents a unique game fleet.

* **Example Git Directory Structure for Games:**  
  manifests/  
  └── games/  
   ├── minecraft-free/  
   │ └── fleet.yaml  
   ├── minecraft-member/  
   │ └── fleet.yaml  
   └── ark-member/  
   └── fleet.yaml
* An ApplicationSet resource will be configured to watch this manifests/games/ directory. For each subdirectory it finds, it will use a template to generate a new ArgoCD Application. The name of the subdirectory can be used as a parameter in the template to name the application and define other metadata.39

This design creates a powerful, scalable workflow for managing the platform's core offering. Adding support for a new game (e.g., "Factorio - Free Tier") becomes as simple as creating a new factorio-free/ directory with its corresponding Agones fleet.yaml manifest and pushing it to Git. The ApplicationSet controller will automatically detect the new directory and instruct ArgoCD to deploy the new fleet, with zero manual intervention required. This is a massive operational win that is essential for enabling the platform to scale its game library while maintaining a lean engineering team.

## **5. Project Structure & Naming Conventions**

### **5.1 Introduction**

Consistency is a key principle of operational excellence. A clear and enforced set of naming conventions and a logical project structure for Kubernetes manifests are essential for maintainability, readability, and reducing cognitive overhead. When another engineer or even the original author returns to the codebase months later, the purpose and relationship of each component should be immediately obvious. This section defines the standards for the "We The Gamers" platform.

### **5.2 Naming Conventions for Kubernetes Objects**

All Kubernetes resources created will adhere to a consistent app-component-tier format, where applicable. This convention provides clarity at a glance.

* **General Format:** <app-name>-<component>-<optional-tier>
* **Examples:**
  + **Deployments:** frontend-api, discord-bot, agones-controller
  + **Services:** frontend-api-svc, postgres-cluster-primary-svc
  + **ConfigMaps:** frontend-api-config, prometheus-rules-cm
  + **Secrets (Managed by Vault):** While the Kubernetes Secret objects created by the injector will have generated names, the paths within Vault will be structured logically, e.g., secret/data/apps/frontend-api, secret/data/database/postgres-creds.
  + **Namespaces:** web-apps, discord-bot, agones-system, monitoring, database, vault. Namespaces will be used to logically isolate applications and control the scope of RBAC roles and Network Policies.
  + **Labels:** Labels are the primary mechanism for selecting and organizing objects. A standard set of labels will be applied to all workloads:
    - app.kubernetes.io/name: The name of the application (e.g., frontend-api).
    - app.kubernetes.io/instance: A unique instance name (e.g., frontend-api-prod).
    - app.kubernetes.io/version: The application version (e.g., 1.2.3).
    - app.kubernetes.io/component: The component within the architecture (e.g., api, worker, database).
    - app.kubernetes.io/part-of: The parent application or system (e.g., we-the-gamers).
    - wtg.io/tier: A custom label for our service tiers (free, member, platform).

### **5.3 GitOps Manifest Directory Structure with Kustomize**

To manage Kubernetes manifests effectively, especially across different environments (e.g., a future staging environment vs. production), we will use Kustomize. Kustomize allows for a template-free way to customize application configurations, leaving the original YAML manifests untouched.

The project structure for storing manifests will be organized by application, with a base directory for common resources and overlays for environment-specific patches.

* **Directory Structure Example (frontend-api):**  
  manifests/  
  └── apps/  
   └── frontend-api/  
   ├── base/  
   │ ├── deployment.yaml  
   │ ├── service.yaml  
   │ ├── ingress.yaml  
   │ ├── networkpolicy.yaml  
   │ └── kustomization.yaml # Defines the base resources  
   └── overlays/  
   └── production/  
   ├── configmap.yaml # Production-specific config values  
   ├── replicas.yaml # Patch to set replica count  
   └── kustomization.yaml # Points to the base and applies patches
* **base Directory:** Contains the core, environment-agnostic Kubernetes manifests for the application (Deployment, Service, etc.). The Deployment here would use a placeholder for the image tag.
* **overlays/production Directory:** Contains only the *differences* for the production environment. This could include:
  + A kustomization.yaml file that references the base and specifies patches.
  + A patch file to set the number of replicas.
  + A ConfigMap with production-specific environment variables.
  + The image tag for the production deployment.
* **ArgoCD Integration:** The ArgoCD Application manifest for the frontend-api will point directly to the overlays/production directory. ArgoCD has native support for Kustomize and will automatically run kustomize build to generate the final, consolidated manifests before applying them to the cluster.

This structure provides a clean separation of concerns. The base defines the application's architecture, while overlays define its configuration for a specific environment. This makes the system highly maintainable and scalable, allowing for the easy addition of new environments (like staging) without duplicating the core manifests.

## **6. Game Server Management (Multi-Tenant & Tiered)**

### **6.1 Introduction**

The core product of the "We The Gamers" platform is the on-demand provisioning of game servers. This requires a robust, scalable, and secure orchestration layer built specifically for this purpose. Agones, an open-source platform built on top of Kubernetes, is the industry standard for hosting, running, and scaling dedicated game servers.1 It extends the Kubernetes API with custom resources like

Fleet and GameServer, simplifying the complex lifecycle management of stateful game server processes.

### **6.2 Orchestration with Agones**

Agones treats game server containers as first-class citizens within the Kubernetes ecosystem. Instead of managing raw Pods, we will manage GameServer objects. Agones provides a sidecar container that integrates with the game server process via an SDK. This sidecar handles health checking, lifecycle management (e.g., moving from Ready to Allocated), and reporting state back to the Kubernetes API.

* **Fleets:** A Fleet is an Agones resource that functions similarly to a Kubernetes Deployment. It defines a template for a GameServer and manages a set of replicas, ensuring a specified number of game servers are always running and ready for allocation.
* **Allocation:** When a user requests a server via the Discord bot, our backend will not create a pod directly. Instead, it will create a GameServerAllocation resource. The Agones controller intercepts this request and finds a Ready GameServer from the appropriate Fleet, marks it as Allocated, and returns its IP address and port. This process is highly efficient and scalable.

### **6.3 Enforcing Service Tiers at the Infrastructure Level**

The business plan's tiered model (Free vs. Member) must be strictly enforced at the infrastructure level to control costs and ensure quality of service.1 This is achieved by combining several Kubernetes and Agones features within the

Fleet definition for each tier.

#### **6.3.1 Resource Isolation with Node Selectors and Tolerations**

This is the most critical enforcement mechanism. As defined in the Cluster Provisioning section, we have dedicated node groups with specific taints (gameserver-tier=free and gameserver-tier=member). We will use nodeSelector and tolerations in our Fleet specifications to ensure game servers are scheduled onto the correct hardware.

* **Free Tier Fleet Manifest Snippet:**  
  YAML  
  apiVersion: agones.dev/v1  
  kind: Fleet  
  metadata:  
   name: minecraft-free-fleet  
  spec:  
   replicas: 10  
   template:  
   spec:  
   # This toleration allows the pod to be scheduled on nodes with the 'free' taint.  
   tolerations:  
   - key: "gameserver-tier"  
   operator: "Equal"  
   value: "free"  
   effect: "NoSchedule"  
   # This selector ensures the pod is scheduled on a node with the ARM architecture.  
   nodeSelector:  
   "kubernetes.io/arch": "arm64"  
   #... rest of the GameServer spec  
    
  This configuration guarantees that free-tier Minecraft servers will only run on the gameserver-free node group, which is composed of low-cost ARM Spot Instances.13
* **Member Tier Fleet Manifest Snippet:**  
  YAML  
  apiVersion: agones.dev/v1  
  kind: Fleet  
  metadata:  
   name: ark-member-fleet  
  spec:  
   replicas: 5  
   template:  
   spec:  
   # This toleration allows the pod to be scheduled on nodes with the 'member' taint.  
   tolerations:  
   - key: "gameserver-tier"  
   operator: "Equal"  
   value: "member"  
   effect: "NoSchedule"  
   nodeSelector:  
   "kubernetes.io/arch": "arm64"  
   #... rest of the GameServer spec  
    
  This ensures that the higher-resource ARK servers for paying members will only run on the gameserver-member node group, which is composed of stable, performant On-Demand or Reserved Instances.

#### **6.3.2 Resource Consumption with Quotas and Requests/Limits**

To prevent any single game server from consuming excessive resources and impacting other tenants on the same node, we will enforce strict resource management.

* **ResourceQuotas:** A ResourceQuota object will be applied to the namespace where game servers run. This will set aggregate limits on the total amount of CPU and memory that can be consumed by all pods in that namespace.
* **Requests and Limits:** Every GameServer template within a Fleet will have requests and limits defined for its containers.
  + requests: The amount of CPU/memory guaranteed for the pod. This is what the Kubernetes scheduler uses to decide where to place the pod.
  + limits: The maximum amount of CPU/memory the pod is allowed to use. The container runtime will throttle or terminate the process if it exceeds this limit.
* **Tiered Example:**
  + **Free Tier (Minecraft):** requests: { cpu: "500m", memory: "1Gi" }, limits: { cpu: "1", memory: "2Gi" }.
  + Member Tier (ARK): requests: { cpu: "2", memory: "6Gi" }, limits: { cpu: "4", memory: "8Gi" }.  
    These values directly correspond to the service tiers outlined in the business plan.1

#### **6.3.3 Dynamic Scaling with FleetAutoscaler**

To ensure there is always a buffer of ready-to-allocate game servers without over-provisioning and wasting money, we will use the Agones FleetAutoscaler. This resource automatically scales the number of replicas in a Fleet based on demand.

* **Free Tier Policy:** The FleetAutoscaler for the minecraft-free-fleet will be configured with a Buffer policy. For example, it will be set to maintain a buffer of 5 Ready game servers at all times. When a server is allocated, the replica count drops below the buffer, and the autoscaler will automatically scale the fleet up to replenish it. The maxReplicas will be set to a high number, allowing the fleet to scale up significantly during peak hours, constrained only by the capacity of the spot instance node group.
* **Member Tier Policy:** The FleetAutoscaler for member-tier fleets will also use a buffer policy but may have a smaller buffer and a more constrained maxReplicas value, reflecting a more predictable usage pattern and a desire to control costs on the more expensive on-demand nodes.

By combining these Kubernetes-native and Agones-specific controls, the platform can programmatically and automatically enforce the multi-tenant, multi-tiered service model, directly linking the business logic to the underlying infrastructure behavior.

## **7. In-Cluster HA Database Strategy**

### **7.1 Introduction**

While managed database services like Amazon RDS are excellent, the "We The Gamers" platform's extreme cost-optimization mandate and the desire for a unified, Kubernetes-native operational model make running the database *inside* the cluster an attractive option. This approach centralizes all infrastructure management under Kubernetes and can be more cost-effective, provided it is implemented correctly using a robust, production-grade Kubernetes Operator. The platform requires a clustered, highly-available PostgreSQL solution.1

### **7.2 The PostgreSQL Operator: Crunchy Data vs. Zalando**

Several open-source operators exist to manage PostgreSQL on Kubernetes. The two most prominent are from Crunchy Data (PGO) and Zalando.

* **Zalando Postgres Operator:** Known for its simplicity and ease of use. It is a solid choice for developers and mid-sized clusters.40
* **Crunchy Data Postgres Operator (PGO):** Known for its comprehensive feature set, reliability, and enterprise-grade capabilities. It is designed for complex, production workloads and is backed by a company that specializes in PostgreSQL.40

For "We The Gamers," **the Crunchy Data Postgres Operator (PGO) is the recommended choice.** While Zalando's operator is good, PGO's feature set is more aligned with the long-term needs of a production platform. PGO provides declarative high availability, advanced disaster recovery tooling with pgBackRest, automated and safe upgrade management, and robust monitoring integrations out of the box.41 It is a complete, cloud-native Postgres solution designed for GitOps workflows.

### **7.3 Deployment Methodology**

The entire PostgreSQL cluster lifecycle will be managed declaratively via a PostgresCluster Custom Resource (CR).

* **Installation:** The PGO operator will be installed via its Helm chart into the database namespace.
* **Cluster Provisioning:** A single PostgresCluster manifest will be created and stored in our GitOps repository. This manifest will define the entire topology:
  + **High Availability (HA):** The manifest will specify a cluster with at least three instances. PGO will automatically configure one instance as the primary (writer) and the others as synchronous streaming replicas (readers). It uses a distributed consensus system to manage leader election and ensure safe, automated failover. If the primary pod fails, PGO will promote a replica to become the new primary within seconds.41
  + **Scheduling:** The PostgresCluster CR will include pod anti-affinity rules to ensure that no two Postgres instances are scheduled on the same physical node, preventing a single node failure from taking down the entire database. The pods will be scheduled onto the stable core-services node group using node selectors and tolerations.
  + **Resource Allocation:** The pods will run on ARM-based Graviton instances for optimal price-performance.13 Specific CPU and memory requests/limits will be set in the manifest to guarantee resources for the database.
  + **Connection Pooling:** PGO natively integrates PgBouncer, a lightweight connection pooler for PostgreSQL. The manifest will enable PgBouncer, which is crucial for handling a large number of connections from the various platform microservices without overwhelming the PostgreSQL processes themselves. PGO will expose a separate Service for the connection pooler, and all applications will connect to this service endpoint, not directly to the database.

### **7.4 Backup and Disaster Recovery**

PGO's greatest strength is its deep integration with **pgBackRest**, a powerful open-source backup and restore tool for PostgreSQL.41

* **Backup Strategy:** The PostgresCluster manifest will define the backup strategy:
  + **Repository:** Backups will be stored in a dedicated S3 bucket for cost-effective, durable, off-site storage.
  + **Schedule:** pgBackRest will be configured to perform a **full backup** once a week and **incremental backups** daily. This is highly efficient, as incremental backups only store the data that has changed since the last backup.
  + **WAL Archiving:** Continuous Write-Ahead Log (WAL) archiving will be enabled. This is the key to Point-in-Time Recovery (PITR). Every transaction log is archived to S3 as it's generated.
* **Restore Process:**
  + **Point-in-Time Recovery (PITR):** In the event of data corruption or accidental deletion, PGO can orchestrate a PITR. A new PostgresCluster can be created that restores from the latest full backup and then replays the archived WAL files up to a specific timestamp, just before the catastrophic event occurred. This allows for recovery with minimal data loss.
  + **Full Cluster Restore:** In a total disaster scenario, the entire cluster can be recreated from a backup in S3 by simply applying a new PostgresCluster manifest with the appropriate restore configuration.

By using a mature operator like PGO, we can achieve a level of automation, high availability, and disaster recovery for our in-cluster database that rivals managed cloud services, while retaining full control and optimizing for cost within our Kubernetes-native ecosystem.

## **8. Cost Optimization & Management**

### **8.1 Introduction**

The financial viability of the "$0.99 member" model is predicated on a relentless and multi-faceted approach to cost optimization. Every architectural decision must contribute to minimizing operational expenditure. This section details a suite of aggressive, practical strategies to ensure the platform runs as leanly as possible without compromising the core user experience.

### **8.2 Compute Cost Reduction Strategies**

Compute is the single largest component of the cloud bill. The following strategies will be employed in concert to reduce this cost.

* **Extensive Use of Spot Instances:** As detailed in the cluster provisioning section, the gameserver-free node group will run exclusively on AWS Spot Instances. With potential savings of up to 90% compared to on-demand prices, this is the single most impactful cost-saving measure.10 The use of a mixed-instances policy across multiple instance families and sizes maximizes the likelihood of acquiring and retaining spot capacity.
* **ARM Architecture Adoption (Graviton):** All workloads that can be compiled for ARM will be. This includes all core platform services, the database, and as many game servers as possible (starting with Minecraft). The 20-40% improvement in price-performance offered by AWS Graviton instances provides a massive, continuous reduction in compute costs for the entire platform.13
* **Intelligent Node Autoscaling (Cluster Autoscaler & Karpenter):**
  + **Cluster Autoscaler:** The standard Kubernetes Cluster Autoscaler will be used to automatically adjust the number of nodes in our node groups. It will monitor for pods that are in a Pending state due to insufficient resources and add new nodes to the appropriate group. Conversely, it will consolidate pods and terminate underutilized nodes to save money.
  + **Karpenter:** For the gameserver-free spot-based node group, we will deploy Karpenter. Karpenter is an open-source, high-performance cluster autoscaler built by AWS. Unlike the standard Cluster Autoscaler which manages node groups, Karpenter manages nodes directly. It can observe the aggregate resource requests of pending pods and make more intelligent decisions, launching the most optimal and cost-effective instance type from a flexible set of choices (e.g., different sizes, families, and AZs) to fit the workload, often resulting in better bin-packing and lower costs than the traditional approach.
* **Right-Sizing Pods (HPA & VPA):**
  + **Horizontal Pod Autoscaler (HPA):** For stateless workloads like our web APIs, the HPA will be used to automatically scale the number of pod replicas up or down based on real-time CPU or memory utilization metrics collected by Prometheus. This ensures we only run as many replicas as are needed to handle the current traffic load.
  + **Vertical Pod Autoscaler (VPA):** The VPA will be run in "recommendation" mode. It will analyze the historical resource usage of pods and recommend optimal request values. This data is invaluable for right-sizing our pod specifications, preventing us from over-provisioning resources and wasting money.

### **8.3 Resource Governance and Waste Prevention**

* **Strict Resource Quotas and LimitRanges:**
  + **ResourceQuota:** Will be applied to each namespace to set hard caps on the total amount of CPU, memory, and persistent storage that can be consumed. This prevents any single application or a runaway process from consuming all cluster resources.
  + **LimitRange:** Will be applied to each namespace to enforce default resource request and limit values for any pods created without them. This ensures no "naked" pods with unbounded resource consumption can be deployed.
* **Scheduling for Off-Peak Hours:** Non-essential, batch-oriented workloads (e.g., daily analytics jobs, report generation) will be scheduled to run during off-peak hours using Kubernetes CronJobs. We can even configure a CronJob to scale down the replicas of development or staging environments to zero overnight and on weekends, stopping their resource consumption entirely when they are not in use.

By combining these strategies—leveraging the cheapest pricing models (Spot), the most efficient architecture (ARM), and intelligent, automated scaling at both the node and pod level—the platform can aggressively drive down its operational costs to a level that makes the $0.99 business model sustainable.

## **9. Disaster Recovery & Backup**

### **9.1 Introduction**

A comprehensive disaster recovery (DR) plan is essential for any production service. It ensures business continuity in the face of various failure scenarios, from accidental data deletion to a full regional outage. For "We The Gamers," the DR strategy must protect two distinct types of data: the Kubernetes cluster state itself (i.e., all the YAML resources stored in etcd) and the persistent data stored in volumes (PostgreSQL data and saved game worlds). Velero, an open-source tool for safely backing up and restoring resources in a Kubernetes cluster, will be the cornerstone of this strategy.

### **9.2 Velero for Cluster State and Volume Snapshots**

Velero works by querying the Kubernetes API to get the current state of resources and uploading a compressed tarball of the objects to cloud storage, such as an S3 bucket. It can also integrate with the cloud provider's API to trigger snapshots of persistent volumes.43

* **Installation and Configuration:**
  1. Velero will be installed via its Helm chart into its own velero namespace.44
  2. An IAM role with specific permissions to manage S3 buckets and EBS snapshots will be created and associated with the Velero service account using IAM Roles for Service Accounts (IRSA).
  3. A BackupStorageLocation resource will be configured, pointing to a dedicated S3 bucket in our AWS region. This bucket will store the tarballs of our cluster's object definitions.
  4. A VolumeSnapshotLocation resource will be configured for the aws provider, enabling Velero to trigger EBS snapshots.45

### **9.3 Backup Strategy**

A multi-layered backup strategy will be implemented using Velero Schedule resources.

* **Full Cluster State Backup:**
  + **Frequency:** A scheduled backup will run daily.
  + **Scope:** This backup will capture all Kubernetes resources in the cluster (Deployments, Services, ConfigMaps, Agones Fleets, CRDs, etc.), *excluding* the raw data in persistent volumes. This is a lightweight, fast operation.
  + **TTL:** Backups will have a Time-To-Live (TTL) of 30 days, after which Velero will automatically delete the backup resource and its corresponding file in S3.43
* **Persistent Volume Backups:**
  + **Database (PostgreSQL):** The primary backup mechanism for the database will be the pgBackRest process managed by the PGO operator, as detailed in Section 7. This provides the capability for true Point-in-Time Recovery (PITR). However, we will also configure a separate Velero schedule to take a nightly EBS snapshot of the PostgreSQL data volume. This provides a second layer of defense and a simpler (though less granular) block-level recovery option.
  + **Member-Tier Game Servers:** The persistent worlds for paying members are critical data. Velero's file-system backup (FSB) capability, which uses an underlying tool like Restic or Kopia, will be used for this.46 A Velero  
    Schedule will be created that uses a label selector to target all persistent volumes associated with the gameserver-tier=member pods. This schedule will run nightly, backing up the contents of the game world directories from the volume's filesystem directly to the S3 bucket.

### **9.4 Restore and Disaster Recovery Plan**

The restore process varies depending on the nature of the disaster.

* **Scenario 1: Accidental Deletion of a Resource:**
  + **Action:** An administrator accidentally deletes a critical Deployment or Service.
  + **Recovery:** Use the velero restore create --from-backup <daily-backup-name> --include-resources deployments command to restore just the deleted object from the latest daily backup. Recovery time is seconds to minutes.
* **Scenario 2: Persistent Volume Corruption (Member Game Server):**
  + **Action:** A game server world file becomes corrupted.
  + **Recovery:** The user can request a restore via a support ticket. An admin will use the velero restore create command, pointing to the latest nightly file-system backup for that specific PersistentVolumeClaim, to restore the game world to its state from the previous night.
* **Scenario 3: Full Cluster/Region Failure:**
  + **Action:** A catastrophic failure makes the entire EKS cluster or AWS region unavailable.
  + **Recovery:**
    1. Using Terraform, provision a new, identical EKS cluster in a different AWS region.
    2. Install Velero into the new cluster and configure it to point to the S3 backup bucket (which should be replicated across regions or be globally accessible).
    3. Velero's object storage sync feature will automatically detect the backup files in S3 and create the corresponding Backup CRs in the new cluster's API.43
    4. Initiate a full restore using velero restore create --from-backup <latest-full-backup>. Velero will recreate all Kubernetes resources.
    5. For persistent volumes, Velero will use the EBS snapshots and file-system backups stored in S3 to provision and populate new PersistentVolumes in the new region.
    6. Update DNS records to point to the new cluster's Ingress controller.

This comprehensive DR plan, powered by Velero and automated via its CRDs, ensures that the "We The Gamers" platform can recover from a wide range of failures with minimal data loss and downtime.

## **10. Multi-Level Management & Bot-Driven Operations**

### **10.1 Introduction**

The platform's operational model is designed to be managed by a single engineer initially, with routine tasks delegated to non-technical community admins.1 This requires a carefully architected system of abstraction that exposes powerful capabilities through a simple, secure interface—the central Discord bot. This section details how the underlying Kubernetes architecture supports this multi-level administration model.

### **10.2 The Tiers of Administration**

The architecture provides three distinct levels of access and control, corresponding to the different user roles.

* **Superadmin (Lead Engineer):**
  + **Access Method:** Full, direct kubectl access to the cluster, typically via a secure bastion host or VPN connection into the VPC.
  + **Permissions:** Bound to the superadmin ClusterRole, this user has unlimited permissions to perform any action on any resource in the cluster.
  + **Use Cases:** Deep diagnostics (e.g., debugging node issues), performing cluster upgrades, managing the underlying infrastructure via Terraform, and responding to critical incidents that cannot be handled through automation. This level of access is reserved for the most trusted and technically proficient individual.
* **Server Admin (Community Admins):**
  + **Access Method:** Interacting with the "RedBot" in dedicated Discord channels using specific slash commands (e.g., /gameserver\_restart <server-id>, /gameserver\_logs <server-id>).
  + **Permissions:** The community admin has **zero** direct access to the Kubernetes API. Their actions are mediated entirely by the bot.
  + **Use Cases:** Performing routine game server lifecycle management: viewing server logs, restarting a misbehaving server, or terminating a server that is violating community rules.
* **Standard User / Paying User:**
  + **Access Method:** Interacting with the "RedBot" via simple commands or interactive buttons within a Discord embed.
  + **Permissions:** Users can only manage the specific game server instance that has been allocated to them.
  + **Use Cases:** Starting their server, stopping it, or receiving the connection information (IP and port). Paying users will also receive a link to their dedicated web dashboard for more detailed management.

### **10.3 The API Gateway: A Secure Abstraction Layer**

The key to making this model work securely is an API gateway that sits between the Discord bot and the Kubernetes API. The Discord bot itself will not contain any Kubernetes credentials or logic.

* **Workflow for a Server Admin Command:**
  1. A Server Admin types /gameserver\_restart server-123 in Discord.
  2. The Discord bot receives this event. It performs initial validation (e.g., confirms the user has the "Server Admin" Discord role).
  3. The bot makes a secure, authenticated REST API call to our internal API Gateway endpoint, e.g., POST /api/v1/admin/servers/server-123/restart.
  4. The API Gateway receives the request. It authenticates the bot (e.g., using a pre-shared API key stored in Vault).
  5. The API Gateway, which is running as a pod in the cluster, uses its own Kubernetes Service Account to interact with the Kubernetes API.
  6. This Service Account is bound to the highly restrictive server-admin Role defined in the RBAC section.
  7. The gateway translates the REST call into the appropriate Kubernetes API action. For a restart, this might involve patching the GameServer object to trigger a shutdown and then creating a new GameServerAllocation request.
  8. The Kubernetes API server validates that the gateway's Service Account is authorized by the server-admin RBAC role to perform this action.
  9. The action is executed, and the result is passed back up the chain to the user in Discord.

This architecture creates a vital security boundary. The Discord bot is a publicly exposed application; it should be treated as untrusted. The API Gateway is an internal, trusted service. By mediating all requests through this gateway, we ensure that all actions against the cluster are performed by a single, known identity (the gateway's Service Account) whose permissions are surgically limited by RBAC. This prevents any possibility of a compromised bot or a malicious community admin from performing unauthorized actions on the cluster. Tools like Botkube offer integrations that can provide a similar chat-based interaction model with Kubernetes, demonstrating the feasibility and power of this ChatOps approach.47

## **11. Comprehensive Observability Stack**

### **11.1 Introduction**

Observability—comprising metrics, logs, and traces—is not a luxury; it is a fundamental requirement for operating a reliable and performant system. A comprehensive observability stack allows engineers to understand the health of the platform, diagnose problems, and gain insights into resource consumption. For "We The Gamers," this stack must be cost-effective, powerful, and capable of providing tailored views for different administrative roles. The chosen stack will be the kube-prometheus-stack for metrics and the Loki stack for logging.

### **11.2 Monitoring with Prometheus and Grafana**

* **What it is:** The kube-prometheus-stack is a popular Helm chart that deploys a complete, pre-configured monitoring solution. It includes Prometheus for metrics collection and storage, Alertmanager for handling alerts, and Grafana for visualization, along with various exporters like kube-state-metrics and node-exporter to gather detailed metrics from the cluster and its nodes.49
* **Why it's important:** It provides immediate, out-of-the-box visibility into the health and performance of every component in the Kubernetes cluster, from node CPU usage to pod memory consumption and API server latency.
* **Actionable Best Practices:**
  1. The kube-prometheus-stack will be installed via its Helm chart into the monitoring namespace on the core-services node group.
  2. Prometheus will be configured with persistent storage using an EBS volume to ensure metrics are retained across pod restarts. Data retention will be configured for 15 days to balance historical analysis with storage costs.49
  3. Prometheus will be configured to automatically discover and scrape metrics from:
     + All standard Kubernetes components.
     + The Agones controller, which exposes a rich set of metrics about fleets, game server states, and allocations (e.g., agones\_gameservers\_total, agones\_fleet\_replicas\_count).51
     + Our custom applications, which will be instrumented to expose their own metrics in the Prometheus format.

### **11.3 Logging with Loki and Promtail**

* **What it is:** Loki is a horizontally scalable, highly available, multi-tenant log aggregation system developed by Grafana. It is designed to be very cost-effective and is inspired by Prometheus, but for logs. Promtail is the agent that collects logs and ships them to Loki.52
* **Why it's important:** Centralized logging is essential for debugging. Instead of SSH-ing into nodes or running kubectl logs on individual pods, all logs are aggregated into a single, searchable database. Loki's design, which indexes only the metadata (labels) rather than the full text of the logs, makes it significantly cheaper to run and store data than other solutions like Elasticsearch.
* **Actionable Best Practices:**
  1. Loki will be deployed as a stateful application in the monitoring namespace, using S3 for long-term storage of log chunks (the "boltdb-shipper" index type).
  2. Promtail will be deployed as a DaemonSet, ensuring it runs on every node in the cluster. It will automatically discover all running pods on its node, scrape their logs from the filesystem, enrich them with Kubernetes labels (like pod, namespace, app), and forward them to Loki.53

### **11.4 Tiered Dashboards in Grafana**

Grafana will serve as the single pane of glass for all observability data. A series of role-specific dashboards will be created to provide tailored views of the system.

* **Superadmin Dashboard:**
  + **Purpose:** A comprehensive, top-level view of the entire platform's health for the lead engineer.
  + **Data Sources:** Prometheus.
  + **Key Panels:**
    - Cluster-wide resource utilization (CPU, memory, disk).
    - Node status and health (from node-exporter).
    - Kubernetes API server health (latency, error rates).
    - Health and performance of core services (Vault, ArgoCD, PGO).
    - Agones controller metrics (total game servers, allocations, fleet statuses).51
    - Alertmanager firing alerts.
* **Server Admin Dashboard:**
  + **Purpose:** An aggregated, high-level view for community admins to understand the overall state of the game server ecosystem without exposing sensitive cluster details.
  + **Data Sources:** Prometheus.
  + **Key Panels:**
    - Total active game servers (free vs. member).
    - Total allocated vs. ready game servers.
    - Aggregate resource usage across all game servers.
    - Game server allocation success/failure rate.  
      This dashboard will use PromQL queries that aggregate data across fleets (e.g., sum(agones\_gameservers\_total{state="Allocated"}) by (tier)), deliberately hiding per-node or per-pod details.
* **Paying User Web Dashboard:**
  + **Purpose:** A dedicated, embeddable dashboard for paying members to monitor their own game server.
  + **Data Sources:** This will be powered by a custom API endpoint, not direct Grafana access. The backend API will query Prometheus on the user's behalf for their specific game server pod.
  + **Key Panels:**
    - Real-time CPU Usage (gauge).
    - Real-time Memory Usage (gauge).
    - Historical CPU/Memory usage over the last 24 hours (graph).
    - Player Count (if the game server exposes this as a metric).  
      This architecture provides powerful, role-appropriate observability for all stakeholders, from the engineer managing the infrastructure to the player using the service.

## **12. Conclusion**

This architectural blueprint provides a comprehensive and definitive plan for building the "We The Gamers" platform on a foundation of Kubernetes. The design is meticulously crafted to address the project's unique and challenging constraints: extreme cost-efficiency driven by a $0.99 membership model, the need for high automation to support a single-engineer operation, and the requirement for secure delegation of tasks to non-technical community administrators.

The key architectural decisions and their synergistic effects are summarized as follows:

1. **A Cost-Optimized Cloud Foundation:** The selection of **AWS** as the cloud provider is a strategic decision based on its superior spot instance model, which is critical for the free tier's financial viability; its leadership in ARM-based Graviton compute, which provides a sustained 20-40% cost reduction on the platform's largest expense; and its more generous egress traffic free tier. These factors combine to create a more cost-effective foundation for the long term than competitors.
2. **Infrastructure as Code and GitOps:** The entire ecosystem will be provisioned and managed via **Terraform** and **ArgoCD**. This pure IaC and GitOps workflow is the cornerstone of the automation strategy, enabling reliable, repeatable, and auditable management of the entire platform from a single source of truth. The use of the **ApplicationSet controller** further enhances this by automating the management of game server fleets, allowing the platform's offerings to scale with minimal engineering overhead.
3. **A Tiered and Secure Architecture:** The business model's service tiers are physically enforced at the infrastructure level through a multi-tiered **node group architecture** using Kubernetes taints and tolerations. This programmatically isolates free-tier workloads on low-cost spot instances and reserves stable, performant nodes for paying members. This is layered with a zero-trust security posture, including a **default-deny network policy**, enforcement of the **restricted Pod Security Standard**, and centralized secrets management with **HashiCorp Vault**.
4. **Secure Delegated Administration:** The platform safely empowers community admins through a system of **abstraction and least privilege**. A highly restrictive **RBAC role** is combined with a secure **API gateway**, creating a chokepoint that allows admins to perform routine tasks via a Discord bot without ever gaining direct or dangerous access to the underlying cluster.
5. **Cloud-Native Service Delivery:** By leveraging best-in-class open-source projects like **Agones** for game server orchestration, the **Crunchy Data Postgres Operator** for in-cluster high-availability databases, and the **kube-prometheus-stack** for comprehensive observability, the platform is built entirely on a modern, Kubernetes-native foundation. This ensures a cohesive, maintainable, and scalable system.

In conclusion, this blueprint does not merely describe a collection of technologies; it details an integrated system where each component is chosen and configured to directly support the core business and operational requirements. By adhering to this plan, "We The Gamers" can build a robust, secure, and hyper-efficient technical platform capable of delivering on its community-centric vision and achieving financial sustainability, even with its ambitious low-cost model.

#### Works cited

1. We The Gamers: A Business Plan for a Community-Centric Gaming Platform
2. Kubernetes in the Cloud: AWS EKS vs Azure AKS vs GCP GKE - DEV Community, accessed on June 18, 2025, <https://dev.to/mechcloud_academy/kubernetes-in-the-cloud-aws-eks-vs-azure-aks-vs-gcp-gke-3akj>
3. EKS Pricing: A Complete Breakdown (2025 Guide) - DevZero, accessed on June 18, 2025, <https://site.devzero.dev/blog/eks-pricing>
4. EKS Pricing In 2025: Understand And Optimize Your EKS Costs - CloudZero, accessed on June 18, 2025, <https://www.cloudzero.com/blog/aws-eks-pricing/>
5. Kubernetes Cost: EKS vs AKS vs GKE - Sedai, accessed on June 18, 2025, <https://www.sedai.io/blog/kubernetes-cost-eks-vs-aks-vs-gke>
6. Understanding AKS Pricing: Key Cost Drivers Explained - Cloudchipr, accessed on June 18, 2025, <https://cloudchipr.com/blog/aks-pricing>
7. AWS vs GCP vs Azure: Cloud Credits, Kubernetes, Multi-Cloud Deployment, and User Ratings | Kapstan, accessed on June 18, 2025, <https://www.kapstan.io/blog/aws-vs-gcp-vs-azure-cloud-credits-kubernetes-multi-cloud-deployment-and-user-ratings>
8. Azure Kubernetes Service (AKS) pricing, accessed on June 18, 2025, <https://azure.microsoft.com/en-us/pricing/details/kubernetes-service/>
9. Cloud Pricing Comparison: AWS vs. Azure vs. Google Cloud Platform in 2025 - Cast AI, accessed on June 18, 2025, <https://cast.ai/blog/cloud-pricing-comparison/>
10. Understanding Spot Instances Across AWS, Google Cloud, and Azure - ProsperOps, accessed on June 18, 2025, <https://www.prosperops.com/blog/spot-instances/>
11. AWS vs. Azure vs. Google: Compute Pricing Comparison | News - Essential Designs, accessed on June 18, 2025, <https://www.essentialdesigns.net/news/aws-vs-azure-vs-google-compute-pricing-comparison>
12. GCP releases Spot VMs, the next generation of Pre-emptible VMs | Hacker News, accessed on June 18, 2025, <https://news.ycombinator.com/item?id=28847909>
13. ARM vs x86: The Complete 2025 Cloud Performance and Cost Analysis Guide | sanj.dev, accessed on June 18, 2025, <https://sanj.dev/post/arm-vs-x86-cloud-2025>
14. GCP, AWS, Azure, and OCI ARM-Based Server Performance Comparison | Apache APISIX, accessed on June 18, 2025, <https://apisix.apache.org/blog/2022/08/12/arm-performance-google-aws-azure-with-apisix/>
15. Google Follows Suit With Microsoft On Ampere Arm Instances - The Next Platform, accessed on June 18, 2025, <https://www.nextplatform.com/2022/07/15/google-follows-suit-with-microsoft-on-ampere-arm-instances/>
16. GCP, AWS, and Azure ARM-Based Server Performance Comparison - API7.ai, accessed on June 18, 2025, <https://api7.ai/blog/arm-performance-google-aws-azure-with-apisix>
17. Deliver better price-performance by selecting the right cloud to run APISIX API gateway, accessed on June 18, 2025, <https://blogs.oracle.com/cloud-infrastructure/post/select-right-cloud-for-apisix-api-gateway>
18. Did AWS (Azure and GCP) started as a cheap way for cloud or just convenience? - Reddit, accessed on June 18, 2025, <https://www.reddit.com/r/ExperiencedDevs/comments/1jglb2z/did_aws_azure_and_gcp_started_as_a_cheap_way_for/>
19. AWS Egress Costs in 2025: How to Reduce Them? - nOps, accessed on June 18, 2025, <https://www.nops.io/blog/aws-egress-costs-and-how-to-avoid/>
20. Cloud Pricing Comparison 2025: AWS vs. Azure vs. Google Cloud Blog - Aress Software, accessed on June 18, 2025, <https://www.aress.com/blog/read/cloud-pricing-comparison-aws-vs-azure-vs-google-cloud>
21. Cloud Pricing Comparison 2025: Compute, Storage, and Networking | emma Blog, accessed on June 18, 2025, <https://www.emma.ms/blog/cloud-pricing-comparison-compute-storage-and-networking>
22. EKS Best Practices to Secure your SaaS Product - ClickIT, accessed on June 18, 2025, <https://www.clickittech.com/cloud-services/eks-best-practices/amp/>
23. terraform-aws-modules - eks, accessed on June 18, 2025, <https://registry.terraform.io/modules/terraform-aws-modules/eks/aws/7.0.0>
24. Taints and Tolerations | Kubernetes, accessed on June 18, 2025, <https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/>
25. Configure workload separation in GKE | Google Kubernetes Engine (GKE), accessed on June 18, 2025, <https://cloud.google.com/kubernetes-engine/docs/how-to/workload-separation>
26. Provisioning AWS EKS Cluster with Terraform - Tutorial - Spacelift, accessed on June 18, 2025, <https://spacelift.io/blog/terraform-eks>
27. Vault installation to Amazon Elastic Kubernetes Service via Helm - HashiCorp Developer, accessed on June 18, 2025, <https://developer.hashicorp.com/vault/tutorials/kubernetes-platforms/kubernetes-amazon-eks>
28. Kubernetes - Auth Methods | Vault - HashiCorp Developer, accessed on June 18, 2025, <https://developer.hashicorp.com/vault/docs/auth/kubernetes>
29. Kubernetes Auth Method (API) - Vault - HashiCorp Developer, accessed on June 18, 2025, <https://developer.hashicorp.com/vault/api-docs/auth/kubernetes>
30. Vault Agent Injector - HashiCorp Developer, accessed on June 18, 2025, <https://developer.hashicorp.com/vault/docs/deploy/kubernetes/injector>
31. Vault Agent Injector examples - HashiCorp Developer, accessed on June 18, 2025, <https://developer.hashicorp.com/vault/docs/deploy/kubernetes/injector/examples>
32. Kubernetes RBAC: A Step-by-Step Guide for Securing Your Cluster - Trilio, accessed on June 18, 2025, <https://trilio.io/kubernetes-best-practices/kubernetes-rbac/>
33. Kubernetes RBAC and Role Aggregation Made Easy - DEV Community, accessed on June 18, 2025, <https://dev.to/hkhelil/kubernetes-rbac-and-role-aggregation-made-easy-3j4o>
34. Global default deny policy best practices - Calico Documentation, accessed on June 18, 2025, <https://docs.tigera.io/calico-cloud/network-policy/default-deny>
35. Policy best practices - Calico Documentation, accessed on June 18, 2025, <https://docs.tigera.io/calico-cloud/network-policy/policy-best-practices>
36. Getting Started - Argo CD - Declarative GitOps CD for Kubernetes, accessed on June 18, 2025, <https://argo-cd.readthedocs.io/en/stable/getting_started/>
37. Getting Started :: ArgoCD Tutorial, accessed on June 18, 2025, <https://redhat-scholars.github.io/argocd-tutorial/argocd-tutorial/02-getting_started.html>
38. Introduction to ApplicationSet controller - Argo CD - Read the Docs, accessed on June 18, 2025, <https://argo-cd.readthedocs.io/en/stable/operator-manual/applicationset/>
39. Templates - Argo CD - Declarative GitOps CD for Kubernetes, accessed on June 18, 2025, <https://argo-cd.readthedocs.io/en/stable/operator-manual/applicationset/Template/>
40. A Comparison of Postgres Operators for Kubernetes Clusters - Akmatori Blog, accessed on June 18, 2025, <https://akmatori.com/blog/a-comparison-of-postgres-operators>
41. CrunchyData/postgres-operator: Production PostgreSQL for Kubernetes, from high availability Postgres clusters to full-scale database-as-a-service. - GitHub, accessed on June 18, 2025, <https://github.com/CrunchyData/postgres-operator>
42. Crunchy Postgres For Kubernetes, accessed on June 18, 2025, <https://www.crunchydata.com/products/crunchy-postgresql-for-kubernetes>
43. How Velero Works, accessed on June 18, 2025, <https://velero.io/docs/v1.4/how-velero-works/>
44. Run Velero on AWS, accessed on June 18, 2025, <https://velero.io/docs/v1.0.0/aws-config/>
45. Run Velero on AWS, accessed on June 18, 2025, <https://velero.io/docs/v0.11.0/aws-config/>
46. File System Backup - Velero Docs, accessed on June 18, 2025, <https://velero.io/docs/main/file-system-backup/>
47. Discord & Botkube Kubernetes Integration, accessed on June 18, 2025, <https://botkube.io/integration/discord>
48. Discord - Botkube, accessed on June 18, 2025, <https://docs.botkube.io/1.13/installation/discord/>
49. Comprehensive Guide to Kube Prometheus Stack with Helm - bootvar, accessed on June 18, 2025, <https://bootvar.com/kube-prometheus-stack-explained/>
50. kube-prometheus-stack 56.6.2 - Artifact Hub, accessed on June 18, 2025, <https://artifacthub.io/packages/helm/kube-prometheus-stack-oci/kube-prometheus-stack/56.6.2>
51. Metrics | Agones, accessed on June 18, 2025, <https://agones.dev/site/docs/guides/metrics/>
52. Loki + Grafana + Promtail: Quickstart with Docker Compose - Kubernetes Training, accessed on June 18, 2025, <https://kubernetestraining.io/blog/loki-grafana-promtail-quickstart-with-docker-compose>
53. Promtail agent | Grafana Loki documentation, accessed on June 18, 2025, <https://grafana.com/docs/loki/latest/send-data/promtail/>
54. Agones GameServers | Grafana Labs, accessed on June 18, 2025, <https://grafana.com/grafana/dashboards/20213-agones-gameservers/>