Onboarding Document

Knowns:

- Developed in C#
- Team has been solely focussing on application features
- Application runs on local machines
- Application expected to handle millions of active players per day
- Application mandated to run on Kubernetes
- It’s only been tested in isolated development environments
- Post-Launch, team is expected to independently manage application lifecycle

---

Phase 1: Planning and Initial Bootstrap

- **Initial assessment** of the app is completed
- I am aware of **resource requirements[CPU, Memory, SSD I/O, NW bandwidth], target regions**. Resource reservations and quota securing requests have already been made.
- **Networking configuration**, VPCs, subnets and IP address space allocations, ACLs, firewall rules in place.
- **Storage Solutions**: Set up buckets for storing cores, terraform states, assets, binaries.
- **Access Control and IAM configurations** implemented to ensure the right people have access to the right dashboards, tools and environments.
- **Meeting #1 :**
  - **Introduce containerization**: Guide them on how to create **Dockerfiles** optimized for production.
  - Walk them through **Kubernetes fundamentals**: Pods, Deployments, Services, ConfigMaps, Secrets.
- Containerize the app, upload the image to a private image repository
- **Meeting #2**:
  - **Introduce IaC concepts** and explain benefits
  - Walk through a sample and show savings in time and toil
- Infrastructure as Code : source controlled configuration code written in terraform (with Terragrunt wrapper) and its deployment automation being actively worked on.
- **Kubernetes manifests and scaling strategy** is actively being worked on, with results from **stress tests** instrumented into the configuration
- Relevant n**amespaces, environments** have been set up to support QA and stress testing.

- **Meeting # 3:**
  - Introduce **CI/CD** with the chosen tool

  - Show sample jobs and the time saved, toil avoided
- Work has begun on setting up **CI/CD Pipelines** for **build, push-to-dev env, QA, push to prod env**
- **Meeting #4:**
  - Introduce **Auto-Sync and Rollback** mechanism
  - Schedule a **stress test** to demonstrate a real-world scenario
  - **Scaling strategy** - whether to use HPA or scale the cluster with more worker nodes.
  - Introduce database management concepts and best practices - **connection pooling, sharding and redundancy**
- Work has begun to set up **contingency plans and rollback options**
- **Meeting #5:**
  - **Introduce Monitoring** and show sample metrics being instrumented
  - **Introduce Logging** and show how to triage issues
  - **Introduce Alerting** and show how to set thresholds
  - **Introduce Visualizations** and show how to build dashboards, write queries
- Work has begun to **instrument logging and mechanism to set up alerts.**
- Work has begun to **identify metrics, workshop visualizations and dashboards**
- **Meeting #6:**
  - Introduce **automated management** of certificates, secrets
  - Introduce **load-balancing concepts** and what HA looks like for our application
- Requests for **certificate renewals, DNS changes, load-balancer and HA requirements**
- **Meeting #7:**
  - Invite security expert and demonstrate the effects of lax security
  - Provide samples of good security practices
- **Security analysis of application** conducted by Security Team

Phase 2: 1 Month before launch

- **CI/CD already set up** for build, push and kubernetes deployment
- **Monitoring, Logging and Alerting** tools are already instrumented
- **Visualizations, dashboards** have been set up for specific audiences
- Submit **cost forecast** report to the executive board to keep them informed
- Ensure **20-30% burst capacity** of launch-limits to ensure we have room to scale.
- **Alert thresholds** codified, escalation policies set up
- Additional **firewall rules, subnets and other network policies** accounted for and implemented.
- **Additional stress tests on infra-scaling** (cluster autoscaler or terraform based manual scaling)
- **App SLOs** implemented: **Availability, Latency, Error Rate**
- Player facing SLOs identified and queries written to instrument the SLIs.

Phase 3: Two Weeks before launch

- **Wind down stress tests**
- Schedule **code-freeze**
- On-call rotation schedules codified.
- Plans for emergency patches in place.

Phase 4 : Launch Week

- War Room presence
- Serve as **primary-on-call** and point of contact between dev team and infrastructure team
- Have a member of the dev team **shadow as secondary on-call**
- Jump into outages and incidents to triage and bring the service back to healthy state
- Keep stakeholders informed with full transparency of impact, and time-to-resolve
- Prioritize people’s health and wellbeing – launch weeks can be stressful but don’t need to be. Creating an environment of safety and excitement is of utmost importance.

Phase 5 : Post Launch Tasks

- **Generate launch report** : traffic, deployments, revenue, engagement
- **Blameless Post Mortem** on any incidents: what went well, what can be improved on.
- **Office hours** for additional support
- Identify and codify process for - **hotfixes, patching, seasonal-time based events**

Documentation & Runbooks:

- Create **on-call playbooks** for common failures.
- Document **CI/CD workflows and Kubernetes manifests**.
- Provide **troubleshooting guides** for scaling and performance issues.

Training & Hands-On Workshops:

- Conduct **Operations training** for developers.
- Assign **Kubernetes ownership roles** within the team.
- Encourage **developers to monitor dashboards & logs** proactively.
- Encourage **observability-driven development**.
- Establish a **blameless postmortem culture** for outages.

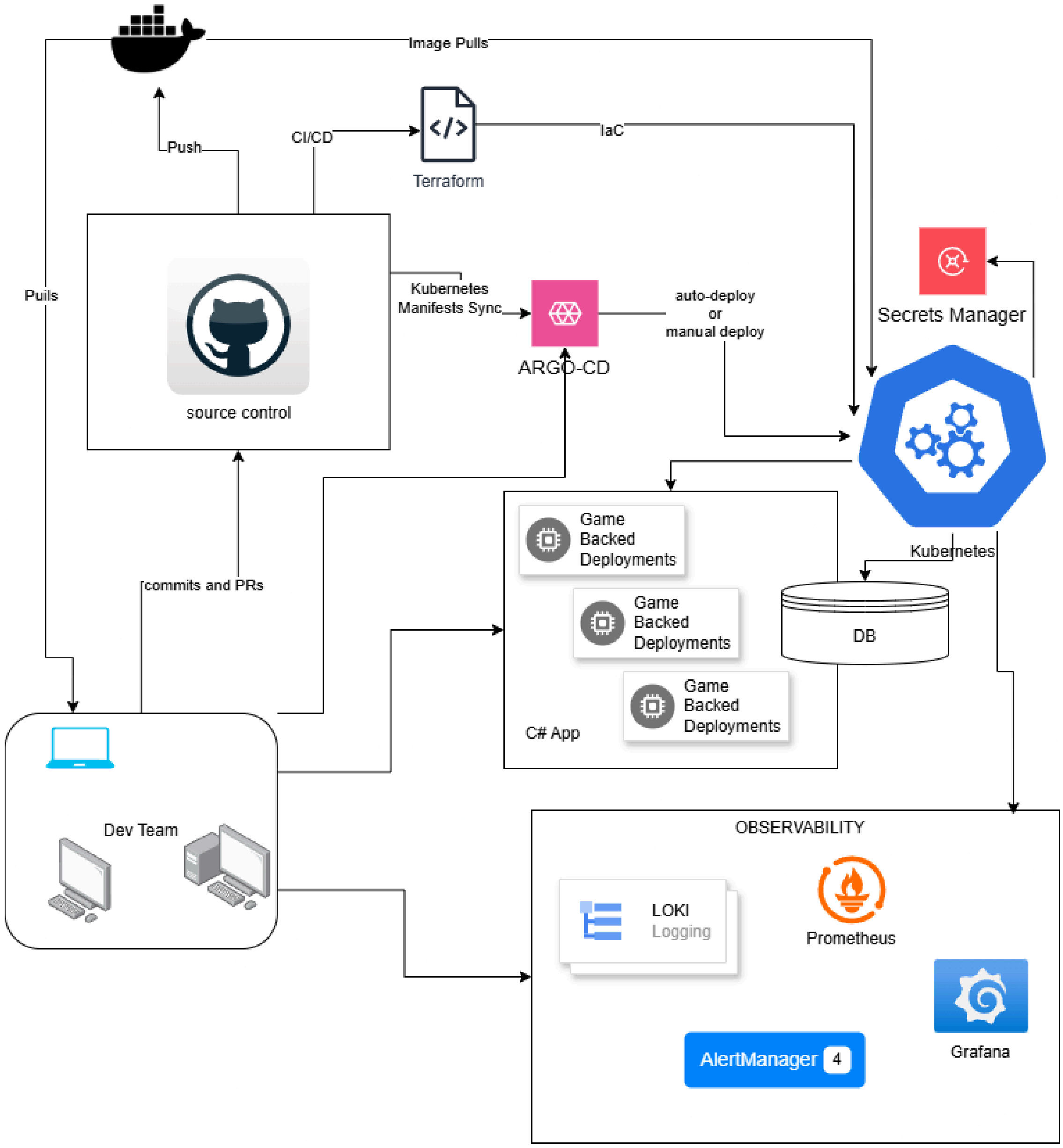
Image Pulls
Push
CI/CD
Terraform
IaC
Puils
source control
Kubernetes
Manifests Sync
ARGO-CD
auto-deploy
or
manual deploy
Secrets Manager
Kubernetes
Game
Backed
Deployments
Game
Backed
Deployments
Game
Backed
Deployments
DB
C# App
commits and PRs
Dev Team
OBSERVABILITY
LOKI
Logging
Prometheus
Grafana
AlertManager 4

Production Readiness Checklist

Item	Tools/Language Used	Status	Notes
Application Code	C# - DotNet	In Use	Provided
Source Control	Github	In Use	Assumption
Unit Testing	Dotnet test and xUnit	In Use	Assumption
Local Build	Dotnet CLI	In Use	Assumption
Build Pipeline	Github Actions	NA	Chosen for exercise
Push Pipeline	Github Actions	NA	Chosen for exercise
Image Repository	Docker-Hub	NA	Chosen for exercise
Image Testing	Trivy and Snyk	NA	Chosen for exercise
Infra Node SKU	GCP (N2, C2, M2)	NA	TBD
Infrastructure as Code	Terraform with Terragrunt wrapper	NA	Chosen for exercise
Kubernetes	GKE	NA	Chosen for exercise
Manifest Management	Helm	NA	Chosen for exercise
Continuous Delivery	ArgoCD	NA	Chosen for exercise
Secrets Management	Vault or External Secrets Operator	NA	Chosen for exercise
Certificate Management	Cert-Manager Helm	NA	Chosen for exercise Assumption: company has a Certificate Authority and ACME protocol compliant
Scaling	HPA and Cluster Autoscaling	NA	Based on CPU and Memory % targets
Stress Testing	In house tool/QA/Simulations	In Use	Assumption
Data Cache	Redis	In Use	Assumption
Database	SQL based	NA	Assumption
Load Balancing	GCP loadbalancer (internal/external)	NA	Chosen for exercise

Ingress	NGINX Ingress	NA	TBD
RBAC (SecOps) and IAM	GCP Service Accounts, Users and Groups with right roles, Kube Service Accounts	NA	TBD
Monitoring	Prometheus	NA	Chosen for exercise
Visualization	Grafana	NA	Chosen for exercise
Logging	Loki	NA	Chosen for exercise
Alerting	AlertManager	NA	Chosen for exercise ● Avoid Alert Fatigue ● Sensible Escalation Policies ● I will be primary on-call during launch-week and post launch week ● Dev team member to participate as secondary

1. Initial Assessment

Goals:

- Understand the current state of the application.
- Bridge the knowledge gap between development and DevOps.
- Align development practices with production needs.

Assess the current state of the application:

- Are there any **hardcoded dependencies** (e.g., file-based storage, local memory caches)? - need to move these to arguments that can be read from a config file managed in a configmap
- What are the **CPU and Memory Requirements** for the application? Is it compute optimized or memory optimized?
- What **regions are we targeting to serve** the application from? Is it a single region or multiple regions? - this will help understand compute instance quota determination to avoid stockouts.
- What sort of **network throughpu**t is expected, to serve the amount of traffic?
- What kind of **network considerations**? IP space, firewall rules, SSL certs and DNS entries?
- **Load Balancing and HA requirements** - does this need to be behind an ingress? Are there path based/hostname based rules to apply? Need to define a scaling strategy and keep it consistent with load-balancing approach.
- Are there metrics currently being considered? Have they been instrumented in the app? If so, are they being exposed and ready to be scraped?
- How are **secrets being managed** currently? - introduce **vault** or **external-secrets operator**
- Are there any **service mesh considerations** (especially for multi-region comms) - if so, we'd need to get consul or something similar in each region, so one region can sync with another region.
- Are there any **message bus integrations**? Common ones used are rabbitmq or pulsar.
- Are there any **cronjobs and batch jobs** that need to be in place? Would be better to use kube based manifests for that.
- Are there any **GDPR and PIPL based considerations** to account for before launch? Need to stay in compliance.
- What are the **external service dependencies** (e.g., databases, APIs)? Is the database akin to a memory store or a dedicated SQL/Oracle DB? Is it located in the same datacenter?

Notes

2. Containerization & Deployment Setup

Goals:

- Ensure the application is containerized correctly. Multi stage for a leaner image size.
- Ensure the image can be built and pushed to a private repo.
- Set up Kubernetes deployment manifests using helm charts.
- Implement CI/CD for automated deployment.

Actions:

Containerization:

- Optimize the **Dockerfile** for smaller image size and faster builds.
- Use **multi-stage builds** to reduce the final image footprint.
- Store images in the company's **container registry**.

Kubernetes Deployment Setup:

- Define **Deployment** (replicas, affinity rules).
- Use **ConfigMaps & Secrets** for environment-specific variables.
- Set up **Services** (ClusterIP for internal, LoadBalancer/Ingress for external).
- Enable **horizontal pod autoscaling (HPA)** to scale based on CPU/memory usage.

CI/CD Pipeline:

- Use **GitHub Actions** to automate builds.
- Implement **image vulnerability scanning** before pushing to the registry.
- Automate **Kubernetes deployment** with ArgoCD and Helm.

Notes

3. Scalability & Performance Optimization

Goals:

- Ensure the backend can handle millions of players.
- Optimize request handling and database performance.

Actions:

Load Testing:

- Use **k6 or Locust** to simulate real-world traffic.
- Identify performance bottlenecks before launch.
- Tune database queries and caching strategies.

Scaling Strategy:

- Implement **Horizontal Pod Autoscaler (HPA)**.
- Use **Cluster Autoscaler** to add/remove nodes dynamically.
- Cache frequently accessed data using **Redis**.
- Optimize **gRPC or WebSocket connections** for real-time communication.
- **GCP Node SKU**: Start with **N2** or **C2** instances (e.g., `n2-standard-8` or `c2-standard-8`) with **8-16 vCPUs** and **32-64 GB RAM** per node.
- **Kubernetes Resource Requests**: Set resource requests based on concurrent player count and expected CPU/memory usage (e.g., `2 vCPUs` and `8 GiB` memory per pod).
- **Scaling**: Implement **HPA** to automatically scale pods based on load, and use **Cluster Autoscaler** to manage node scaling.

Database Considerations:

- Ensure **database connection pooling** is efficient.
- Separate **read/write instances** if using PostgreSQL or MySQL.
- Consider **sharding** if a single database instance won't handle the load.

Notes

4. Security & Compliance

Goals:

- Ensure the application is secure and follows company policies.
- Implement necessary authentication and encryption.

Actions:

Secrets Management:

- Use Kubernetes **Secrets / External Secrets Operator** instead of hardcoding credentials.
- Integrate **HashiCorp Vault** or cloud secret managers if needed (if using cloud-secret managers, ensure that it syncs those secrets with kube-secrets so that they can be securely referenced in manifests)

Network Security:

- Restrict inter-service communication using **NetworkPolicies**.
- Use **Ingress controllers** and **Ingress Resources**

RBAC & Least Privilege Access:

- Enforce **Role-Based Access Control (RBAC)** in Kubernetes.
- Ensure the application runs as a **non-root user** in containers.
- Scan images for vulnerabilities using **Trivy or Snyk**.

Notes

5. Monitoring & Incident Response

Goals:

- Provide visibility into application health and performance.
- Set up alerting for proactive issue resolution.

Actions:

Observability Stack:

- Use **Prometheus & Grafana** for real-time metrics.
- Set up **Loki** for centralized logging.
- Use **Jaeger/Tempo** for distributed tracing.

Alerting & Incident Management:

- Define **SLIs & SLOs** for response times, error rates, and latency.
- Integrate **AlertManager** with Slack/PagerDuty for alerts.

6. Knowledge Transfer & Long-Term Ownership

Goals:

- Ensure the development team can manage the infrastructure post-launch.
- Encourage a DevOps culture within the team.

Notes

Final Outcome

By the end of the engagement, the development team will:

- Have a **containerized, scalable, and resilient** backend.
- Use a **fully automated CI/CD pipeline** for deployments.
- Leverage **Kubernetes-native tools** for monitoring, logging, alerting, scaling, and other operational tasks associated.
- Gain **ownership of operational aspects** post-launch.
- Be empowered to manage their infrastructure **without DevOps hand-holding**.