ARCADA

GitOps-driven Game Hosting på Kubernetes

Tobias Eriksson

Lärdomsprov

Informationsteknik

2025

Lärdomsprov

Tobias Eriksson

GitOps-driven Game Hosting på Kubernetes.

Yrkeshögskolan Arcada: Informationsteknik, 2025.

Sammandrag:

Detta arbete beskriver hur spelserver-hosting kan implementeras i en Kubernetes-miljö med hjälp av GitOps-principer. Projektet bygger på Rancher Desktop som lokalt Kubernetes-kluster, tillsammans med ArgoCD för versionshantering och synkronisering av konfigurationer samt Agones för hantering av spelservrar. Syftet med arbetet är att visa hur modern containerteknik och DevOps-metodik kan användas även i mindre projekt och hobbybaserade miljöer, samt att analysera lösningens styrkor och begränsningar.

Miljön installerades deklarativt via en så kallad app-of-apps-struktur där ArgoCD automatiskt distribuerar och uppdaterar alla komponenter utifrån Git-repositoriet. Tre spelservrar användes i projektet: Minecraft, Xonotic och Terraria. Dessa valdes eftersom de har fungerande stöd för Kubernetes och Agones, samt för att det finns tidigare erfarenhet av att installera dem i traditionella miljöer. Samtliga spelservrar kunde köras parallellt i klustret efter mindre konfigurationsjusteringar.

Resultaten visar att Kubernetes och GitOps ger tydliga fördelar gällande reproducerbarhet, struktur och hantering av uppdateringar, men också att den initiala komplexiteten och det krav på förståelse för Kubernetes och dess arbetsmodell gör lösningen mer krävande än traditionell hosting. Jämförelsen visar att Kubernetes lämpar sig bäst när flera servrar ska hanteras eller när man eftersträvar spårbarhet och automatisering. Arbetet visar att teknik som normalt används i stora företagsprojekt även kan ge värde i mindre sammanhang.

Nyckelord:

Kubernetes, ArgoCD, Infrastruktur som kod (IaC), GitOps, deklarativ konfigurering, spelserver-hosting, reproducerbarhet, skalbarhet

Degree Thesis

Tobias Eriksson

GitOps-driven Game Hosting on Kubernetes.

Arcada University of Applied Sciences: Information technology, 2025.

Abstract:

This thesis describes how game server hosting can be implemented in a Kubernetes environment using GitOps principles. The project is based on Rancher Desktop as a local Kubernetes cluster, combined with ArgoCD for version control and configuration synchronization, and Agones for managing game servers. The purpose of the thesis is to demonstrate how modern container technologies and DevOps practices can be applied even in smaller projects and hobby-oriented environments, as well as to analyse the strengths and limitations of such a solution.

The environment was installed declaratively using an app-of-apps structure, where ArgoCD automatically deploys and updates all components based on the Git repository. Three game servers were used in the project: Minecraft, Xonotic, and Terraria. These were chosen because they offer support for Kubernetes and Agones, and because there is previous experience of installing them in traditional environments. All game servers were able to run in parallel in the cluster after minor configuration adjustments.

The results show that Kubernetes and GitOps provide clear advantages in terms of reproducibility, structure, and update management, but also that the initial complexity and the required understanding of Kubernetes and its operating model make the solution more demanding than traditional hosting. The comparison indicates that Kubernetes is best suited when multiple servers must be managed or when traceability and automation are desired. The thesis demonstrates that technologies normally used in large-scale enterprise environments can also provide value in smaller contexts.

Keywords:

Kubernetes, ArgoCD, Infrastructure as Code (IaC), GitOps, declarative configuration, Game server hosting, Reproducibility, Scalability

Innehåll

1 Inledning

Detta projekt undersöker ett alternativt sätt att driftsätta spelservrar genom att använda Kubernetes som grundplattform, i stället för att köra dem direkt på en Linux- eller Windows-server. Fokus ligger på att kombinera Kubernetes med DevOps- och mer specifikt GitOps-praxis för att skapa en skalbar och reproducerbar hosting-miljö för mindre spelserverprojekt.

Projektet använder ArgoCD som GitOps-verktyg för att automatiskt synkronisera all konfiguration från ett Git-repo till Kubernetes-klustret. I repot finns en strukturerad samling av YAML-filer som beskriver det önskade tillståndet för klustrets olika komponenter. Genom att använda Kustomize och ArgoCD som infrastruktur-och konfigurations-som-kod (IaC och CaC) skapas en "single source of truth" där all kluster-, spel- och resurskonfiguration är versionshanterad och lätt att återskapa. Denna struktur möjliggör skalbarhet, undviker onödig repetition och gör det lättare att underhålla flera spelservrar samtidigt.

Kubernetes och GitOps används i dag i många moderna utvecklings- och driftsmiljöer, särskilt när man vill arbeta deklarativt och versionshanterat. I detta projekt undersöks om samma arbetssätt även kan vara värdefullt i mindre sammanhang, såsom hobbyprojekt eller enklare spelservermiljöer. En viktig fråga är därför vilka fördelar och nackdelar GitOps-praxis hämtar till ett sådant mindre projekt, och om de positiva effekterna från företagsmiljöer till exempel spårbarhet, dokumentation som kod och automatiserad konfiguration också kan vara nödvändiga här.

De huvudnyttorna som detta projekt fokuserar på är skalbarhet, reproducerbarhet och automatisering. Huvudnyttorna uppnås genom GitOps-flödet och YAML-konfigurationerna, som är möjliggjorda med Kubernetes, ArgoCD och de andra komponenten i projektet. Tillsammans skapar dessa en robust och modern plattform för game hosting.

Avslutningsvis ger rapporten en översikt över projektets struktur och arbetsgång samt en jämförelse mellan Kubernetes-baserad hosting och traditionell spelserverdrift. Målet är

att visa att tekniker som Kubernetes, DevOps och GitOps inte bara är reserverade för stora företag, utan även kan tillämpas praktiskt och effektivt i små personliga projekt.

1.1 Arbetets typ och avgränsningar

Detta arbete är i första hand ett utvecklings- och implementationsprojekt. Tyngdpunkten ligger på att beskriva och motivera den tekniska lösningen, val av verktyg, arkitektur samt installation och konfiguration av olika komponenter.

Målet är inte att svara på en specifik forskningsfråga genom mätningar eller tester, utan att dokumentera en fungerande lösning och analysera dess styrkor och begränsningar.

2 Bakgrund

Detta kapitel ger en översikt över de centrala tekniker och principer som projektet bygger på. Här förklaras grunderna i containerorkestrering och Kubernetes, samt hur moderna DevOps-metoder som *Infrastructure-as-Code* och GitOps tillämpas. Kapitlet beskriver även hur spelservrar kan köras i Kubernetes med hjälp av Agones, ett ramverk utvecklat för att hantera spelservrar på Kubernetes. Syftet med bakgrunden är att ge läsaren en teknisk och konceptuell grund för att förstå projektets implementation och resultat.

2.1 Containerorkestrering

Containerorkestrering handlar om att automatiskt hantera driften av många containrar i större system. När applikationer växer och består av flera mikrotjänster räcker det inte längre att starta containrar manuellt. En orkestreringsplattform hanterar därför viktiga funktioner som:

- **Resursallokering:** Placera containrar på lämpliga noder i ett kluster.
- **Skalning:** Skala upp när resursanvändningen ökar och skala ned när den minskar.
- **Self-healing:** Starta om containrar som kraschar eller ersätta dem om de blir oresponsiva.

- **Nätverk och service-discovery:** Se till att containrar kan kommunicera med varandra och att tjänster blir åtkomliga.
- **Konfigurationshantering:** Hantera miljövariabler, containerkonfigurering och hemligheter.

Kubernetes är allmänt betraktat som den ledande plattformen för containerorkestrering och beskrivs av Cloud Native Computing Foundation (CNCF) som ett system för att ”automatisera drift, skalning och hantering av containeriserade applikationer” (CNCF, u.å.).

2.2 Kubernetes

Kubernetes är sedan flera år allmänt accepterat som den dominerande standarden för containerorkestrering. Systemet utvecklades ursprungligen av Google, baserat på deras interna plattform Borg, och donerades senare till CNCF (Kubernetes, 2024).

Kubernetes fungerar deklarativt, vilket innebär att utvecklaren beskriver hur applikationen ska köras. Detta skrivs vanligtvis i YAML-filer. När konfigurationen skickas till Kubernetes API-server, ser resten av Kubernetes komponenterna till att automatiskt skapa, starta, skala eller återställa resurser så att systemet matchar den skrivna konfigurationen (Kubernetes Documentation, u.å.-a).

Ett Kubernetes-kluster består i grunden av två typer av noder. En nod består av en maskin eller virtuell maskin, och kan ha olikt resurskrav beroende på storleken av klustret, eller applikationerna man snurrar i containrarna. Det två nodtyperna är: Master-noder vilka snurrar klustrets control plane som ansvarar för att hålla koll på klustrets tillstånd, och worker-noder där applikationerna faktiskt körs. Control plane består av flera centrala delar, bland annat API-servern, schemaläggaren (scheduler) och kontroll-looparna (controllers), som gemensamt ser till att det deklarerade önskade tillståndet i klustret motsvarar det faktiska tillståndet. (Kubernetes Documentation, u.å.-b, u.å.-c, u.å.-d).

Följande resurstyper är centrala för att förstå hur projektets Kubernetes-miljö fungerar. Dessa är de komponenter som faktiskt användes i implementationen.

2.2.1 Pod

En *pod* är den minsta körbara enheten i Kubernetes. Varje spelserver i projektet körs slutligen som en pod, oavsett om den hanteras genom en Fleet, GameServer eller andra resurser. Podden innehåller en eller flera containrar och ansvarar för själva körningen av spelet.

2.2.2 Service

En *service* exponerar en eller flera pods via en stabil nätverksadress. I projektet används services för att möjliggöra anslutning till spelservrarna. En service gör att spelservern kan nås även om dess underliggande pod byts ut eller startas om.

2.2.3 Custom Resource Definitions (CRD)

En *CRD* är inte en resurs i sig, utan ett sätt att utöka Kubernetes med helt egna resurstyper som implementeras av externa system. CRD:s används när en komponent, såsom ArgoCD behöver definiera nya objekt som inte finns i Kubernetes från början.

Kombinationen av dessa resurser gör att Kubernetes kan erbjuda funktioner som ”self-healing”, där trasiga poddar startas om automatiskt, och rullande uppdateringar, där nya versioner distribueras utan dödtid.

I detta projekt används Kubernetes som containerorkestreringsplattform för att köra och hantera spelservrar, där funktioner som deklarativ konfigurering och reproducerbarhet är värdefulla.

2.3 IaC och GitOps

Infrastructure as Code (IaC) och GitOps är centrala begrepp i moderna DevOps-miljöer. DevOps handlar i stort om att knyta ihop utveckling och drift genom gemensamma arbetssätt och automation, medan GitOps kan ses som en konkret arbetsmodell inom DevOps där Git används som sann källa för deklarativ infrastruktur- och applikations-konfiguration (Red Hat, 2025; CNCF, 2021).

Metoderna bygger på idén att systemets infrastruktur och konfiguration ska beskrivas som kod i stället för att hanteras manuellt eller genom ad-hoc-ändringar, vilket gör miljöer mer reproducerbara, förutsägbara och lättare att återställa (Red Hat, 2025; CNCF, 2021). Metoderna är centrala utgångspunkter i detta projekt.

2.3.1 Infrastructure as Code (IaC)

IaC innebär att all infrastruktur som servrar, nätverk, lagring och andra resurser, definieras i filer, vanligtvis YAML eller JSON. Genom att beskriva miljön som kod kan den version hanteras, granskas och uppdateras med samma arbetsmetoder som vanlig programkod. Det gör också att infrastrukturen kan återställas eller reproduceras snabbt vid fel eller förändringar (Red Hat, 2025).

I Kubernetes används IaC naturligt eftersom alla resurser, såsom Pods, Services och andra resurstyper, redan är definierade som deklarativa YAML-filer.

2.3.2 GitOps

GitOps bygger vidare på IaC genom att använda ett Git-repo som den enda sanna källan för hur miljön ska se ut. All konfiguration lagras i repot, och ändringar görs med standard Git arbetsmetoder. Det gör att varje modifiering är spårbar och kan rullas tillbaka vid behov (CNCF, 2021).

Kärnan i GitOps är att automatiserade agenter kontinuerligt jämför det faktiska tillståndet i klustret med det deklarativa tillståndet i Git och ser till att de matchar. Om det uppstår avvikelser, synkroniseras systemet tillbaka till det deklarerade tillståndet i Git, vilket skapar konsistens och minskar risken för fel (CNCF, 2021).

Förutom spårbarhet och reproducerbarhet ger GitOps även en strukturerad arbetsmetod som fungerar lika bra för små hobbyprojekt som i stora företagsmiljöer. Det möjliggör snabb testning, ändrings historik och återställning av miljön vid misstag, egenskaper som är särskilt värdefulla i experimentella eller skalbara miljöer som game hosting.

I detta projekt fungerar GitOps som navet för all konfiguration. Både Kubernetes-resurserna och Agones-specifika spelserverkonfigurationer hanteras via Git-repot, vilket skapar en stabil och reproducerbar process för installation och uppdatering.

2.3.3 ArgoCD som GitOps verktyg

ArgoCD är ett GitOps-verktyg som automatiskt synkroniserar konfigurationer från en Git-repo till ett Kubernetes-kluster. Det bygger på idén att all konfiguration skrivs som kod och att Git-repot alltid innehåller det önskade tillståndet för systemet (ArgoCD Documentation, u.å.).

Den centrala resursen i ArgoCD är en *Application*. En ArgoCD Applikation beskriver vad som ska installeras (t.ex. en katalog i Git-repot), var det ska installeras (namespace och kluster) och hur synkronisering ska ske. När en Applikation är definierad övervakar ArgoCD den konfigurerade katalogen i Git och ser automatiskt till att Kubernetes-klustret matchar det deklarerade tillståndet (ArgoCD Documentation, u.å.).

I detta projekt används ArgoCD Applikationer för att hantera sig själv, Agones-installationen, Shulker-installationen, och alla spelservrar, vilket gör att hela miljön kan uppdateras och återskapas direkt från Git-repot.

2.4 Game server-hosting med Kubernetes

2.4.1 Containeriserade spelservrar

I detta projekt körs spelservrar som containeriserade applikationer i Kubernetes. Varje spelserver använder sig av färdiga container-images, som är lagade specifikt för de spelen som är i frågan. Detta gör att spelservrarna kan köras konsekvent på olika miljöer utan att behöva hantera operativsystemspecifika skillnader. Containrarna definieras med sina portar, miljövariabler och resursgränser i YAML-filer, vilket gör dem lätta att återanvända. Denna containerbaserade modell är särskilt användbar, eftersom en server snabbt kan startas om, ändras eller dupliceras utan att påverka resten av miljön. Det gör även att flera

olika spelservrar kan köras parallellt på samma kluster, utan att deras konfigurationer orsakar problem med varandra.

2.4.2 Agones – ramverket som används

Agones introducerar egna resurstyper (CRD:s), specifikt utvecklade för att hantera spelservrar i Kubernetes, bland annat Fleet, GameServerSet och GameServer (Agones Documentation, u.å.-a).

Den grundläggande resursen i detta projekt är *Fleet*, som fungerar som det huvudobjektet för att definiera en grupp spelservrar. När en Fleet skapas, genererar Agones-kontrollern automatiskt de underliggande resurserna som krävs, såsom GameServerSet och enskilda GameServer-objekt. Dessa resurser skapas och hanteras av Agones baserat på Fleet-konfigurationen och ansvarar till slut för att köra själva podden som kör spelservern (Agones Documentation, u.å.-a; googleforgames/agones, 2018).

I projektets konfiguration räckte det därför att definiera Fleet-objektet. Agones tog hand om resten av livscykeln och genereringen av de andra resurserna automatiskt.

2.4.3 Fördelar med containerorkestrering i detta projekt

Att använda Agones i kombination med Kubernetes ger flera konkreta fördelar i detta projekt:

- **Reproducerbarhet:** Varje spelserverinstans kan återskapas från samma YAML-konfiguration utan flera manuella steg.
- **Skalbarhet:** Kubernetes och Agones konfiguration möjliggör att ändra mängd på spelservar.
- **GitOps-flöde:** Alla ändringar sker via Git, vilket gör hantering konsekvent och stabil.
- **Enhetlig struktur:** Samma metod kan användas för flera olika spelservrar utan förändringar i arkitekturen.

De fördelar gör att även små hobbyprojekt kan använda sig av tekniker som normalt används i större företagsmiljöer.

2.4.4 Begränsningar och erfarenheter i projektet

Med fördelarna finns några begränsningar också, som blev tydliga i arbetet. Installationen krävde mer förberedelser än traditionell game-hosting, eftersom både Kubernetes och Agones har en högre initial konfigurationsnivå. Hela plattformen och dess verktyg hämtar med sig mera komplexitet. För mycket små projekt kan Kubernetes och Agones upplevas som mer avancerade än nödvändigt, särskilt i jämförelse med att helt enkelt starta en spelserver direkt på en fysisk server eller virtuell maskin.

En annan begränsning är att alla spelservrar inte är designade för containerisering. Vissa spel kräver därför skräddarsydda lösningar eller extra justeringar för att fungera väl i Kubernetes, medan andra redan har färdiga och använda open-source-lösningar för både Kubernetes och Agones. Detta gör att mängden arbete och konfigurering mellan implementering av olika spel är hög.

Det finns också en tydlig lärandetröskel. Kubernetes kräver i sig en viss teknisk förståelse, och Agones bygger vidare på denna komplexitet genom att introducera nya resurstyper och begrepp som behöver behärskas för att arbetet ska flyta på effektivt.

Trots dessa begränsningar visar projektet att Kubernetes och Agones är fullt användbara även i mindre sammanhang, särskilt när målet är reproducerbarhet, struktur och en arbetsmodell som följer modern DevOps-praxis.

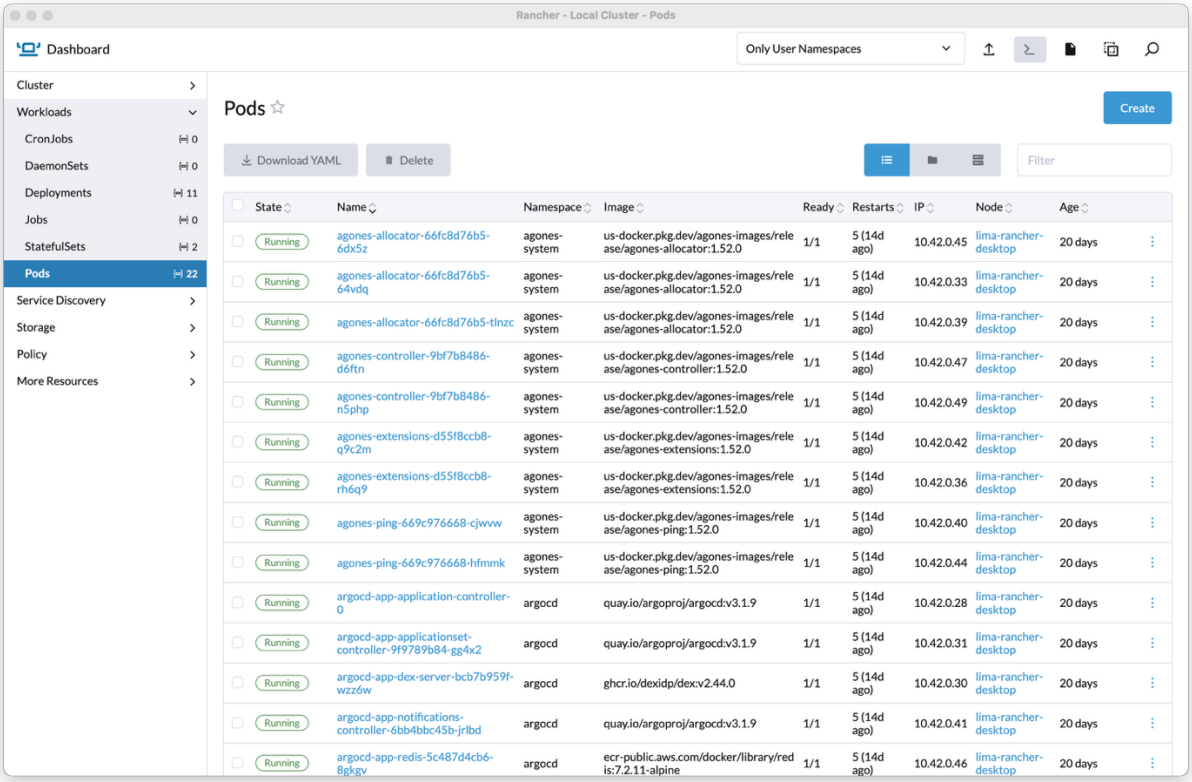
3 Installation

Detta kapitel beskriver installationen och uppsättningen av projektets Kubernetes-miljö, GitOps-flöde och spelservrar. Målet är att ge en reproducerbar och tydlig installationsprocess, med exakta versionsnummer och komponenter som användes under projektets genomförande.

3.1 Val av Kubernetes-miljö

Projektet inleddes med att välja en lämplig lokal utvecklingsmiljö för Kubernetes. Den första tanken var att använda Minikube, som är ett vanligt val för minde Kubernetes miljöer. Efter att ha undersökt olika alertnativ, visade det sig dock att Rancher Desktop var ett bättre alternativ för detta projekt. Rancher Desktop erbjuder en lättanvänd lokal Kubernetes-miljö och har dessutom en inbyggd GUI som liknar den som används i Ranchers enterprise-plattform.

- **Rancher Desktop version:** 1.20.1
- **Kubernetes-version:** 1.33.5



Figur 1. Rancher Desktop GUI

3.2 Installation av grundmiljön

Installationen av projektets grundmiljö började med att ArgoCD installerades direkt i Kubernetes-klustret. Detta gjordes genom att applicera ArgoCD:s officiella installationsmanifest:

```
kubectl apply -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml -n argocd
```

Efter installationen skrevs en dedikerad ArgoCD applikation i projektets Git-repo. Denna resurs möjliggör att ArgoCD kan synkronisera sig själv ("self-managing ArgoCD").
I detta projekt användes:

- **ArgoCD Helm chart version:** 9.0.5
- **ArgoCD applikationsversion:** v3.1.9

Efter cirka tre minuter var ArgoCD färdigt att användas och dess komponenter rapporterade "Running". Nästa steg var att starta upp projektets GitOps-struktur genom att applicera app-of-apps konfigurationen:

```
kubectl apply -f app-of-apps.yaml
```

När *app-of-apps*-resursen applicerades började ArgoCD läsa strukturen i repot, installera Agones via dess Helm-chart, och sätta i gång synkronisering för sig själv också.

3.3 Installation av spelservrar

När grundmiljön var på plats kunde de tre spelservrarna läggas in. Varje spelserver hanterades som en separat modul i Git-repot under `/servers`.

3.3.1 Minecraft

För Minecraft användes det community-drivna projektet Shulker Kubernetes, som lägger till några Kubernetes CRD:s för Minecraft-servrar. Installationen krävde tre komponent:

- En **MinecraftCluster** resurs som definieras för att koppla ihop de nästa två resurser
- En **MinecraftServerFleet** resurs användes för att hantera servergruppen. Den kör i gång en Agones Fleet för servrarna
- En **ProxyFleet** resurs användes som en proxy-server för Minecraft-servern. Denna Custom-Resource är gjord för att kunna ha en ”lobby” server, som kastar spelaren till själva spelservern i Minecraft. Resurs typen är obligatoriskt att använda med Shulker. I vårt fall behövs ingen lobby, så proxy-servern kastar spelaren direkt till spelservern.

När ArgoCD markerade alla objekt som gröna (“Synced” + “Healthy”) testades servern via Minecraft-klient lokalt.

3.3.2 Xonotic

Xonotic användes som exempelspel eftersom det följer med Agones-dokumentationen och är väl anpassat för containerbaserad drift. Konfigureringen blev därför betydligt enklare än för Minecraft. En Agones Fleet-resurs definierades för att hantera servern, och Xonotic-containern startade utan problem då resursen synkroniserades via ArgoCD. Servern exponerades därefter genom en vanlig Kubernetes Service, vilket gjorde den åtkomlig från lokalsystemet. Den enda extra konfigurationen som krävdes var en UDP-portforwarding med socat på lokalsystemet, eftersom Xonotic använder UDP och Kubernetes inte hanterade detta automatiskt. När portforwardingen var på plats kunde spelet anslutas till.

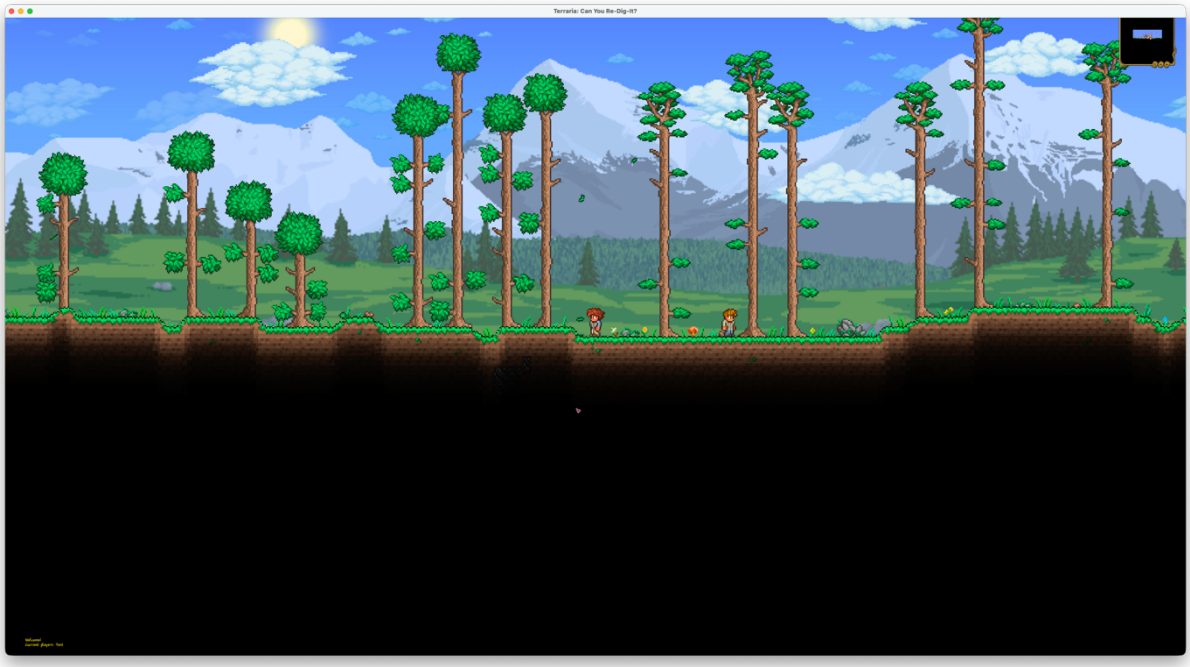
3.3.3 Terraria

Terraria var den enklaste av de tre att installera. En Agones Fleet och en Kubernetes service konfigurerades i Git-repot. Spelet använder TCP, vilket krävde ingen portforwarding. Terraria fungerade utan större hinder direkt efter deployment.

3.4 Verifiering av installationen

När alla komponenter var installerade verifierades installationen på tre nivåer:

1. **GitOps-nivå**
    - ArgoCD visade full synkronisering av alla applikationer
    - Inga “Out of Sync” eller “Unhealthy” resurser
2. **Kubernetes-nivå**
    - Pods listades som ”Running”
    - CRD:s och Custom-Resources från Agones och Shulker hade skapats korrekt
    - Kubernetes Services var ”Ready”
3. **Spel-nivå**
    - Minecraft: klient kunde ansluta
    - Xonotic: klient anslöt över UDP efter portfix
    - Terraria: direkt fungerande anslutning



Figur 2. Terraria spelet anslutet till spel-servern.

3.5 Sammanfattning

Installationen resulterade i en färdig och fungerande Kubernetes-miljö. Rancher Desktop användes som lokalt kluster och gav en stabil grund för projektet. ArgoCD ansvarade för GitOps-flödet och höll all konfiguration synkroniserad med Git-repot. Agones möjliggjorde att spelservrar kunde köras lätt i klustret, och efter installation och mindre konfigurationsjusteringar fungerade alla tre spelservrar. Den slutliga miljön är helt deklarativ, versionerad och reproducerbar, vilket innebär att hela systemet kan återskapas från Git-repot med minimal manuell insats.

4 Implementation och djupare teori

Detta kapitel beskriver hur projektet implementerades i praktiken och går djupare in på de tekniska val, lösningar och arkitektur beslut som formade systemet. Till skillnad från bakgrundskapitlet, som behandlar teknikerna på en teoretisk nivå, fokuserar detta avsnitt på hur Kubernetes, GitOps och Agones integrerades i projektets miljö och hur olika designval påverkade resultatet. Detta kapitel är tekniskt eftersom det binder teori och installation. Det behöver dock inte läsas i detalj för att man ska kunna följa jämförelsen, resultaten eller diskussionen.

4.1 Projektets arkitektur

Projektets arkitektur bygger på tre centrala komponenter som samverkar:

1. **Rancher Desktop** – Lokalt Kubernetes-kluster (v1.33.5)
2. **ArgoCD** – GitOps-verktyg för att synkronisera konfigurationer
3. **Agones** – Ramverk för att hantera spelservrar i Kubernetes

Arkitekturen följer ett deklarativt flöde:
Git-repo → ArgoCD → Kubernetes → Agones → Spelserver-pods

GitHub-repot fungerar som *single source of truth*, ArgoCD övervakar repot och synkroniserar konfigurationer, Kubernetes ansvarar för containerorkestreringen, och Agones hanterar spelserver-specifika resurser som Fleet och GameServer.

4.2 Git-repo för IaC och GitOps

All konfiguration lagras i ett publikt Github-repo:
https://github.com/Tobias-Eriksson/thesis

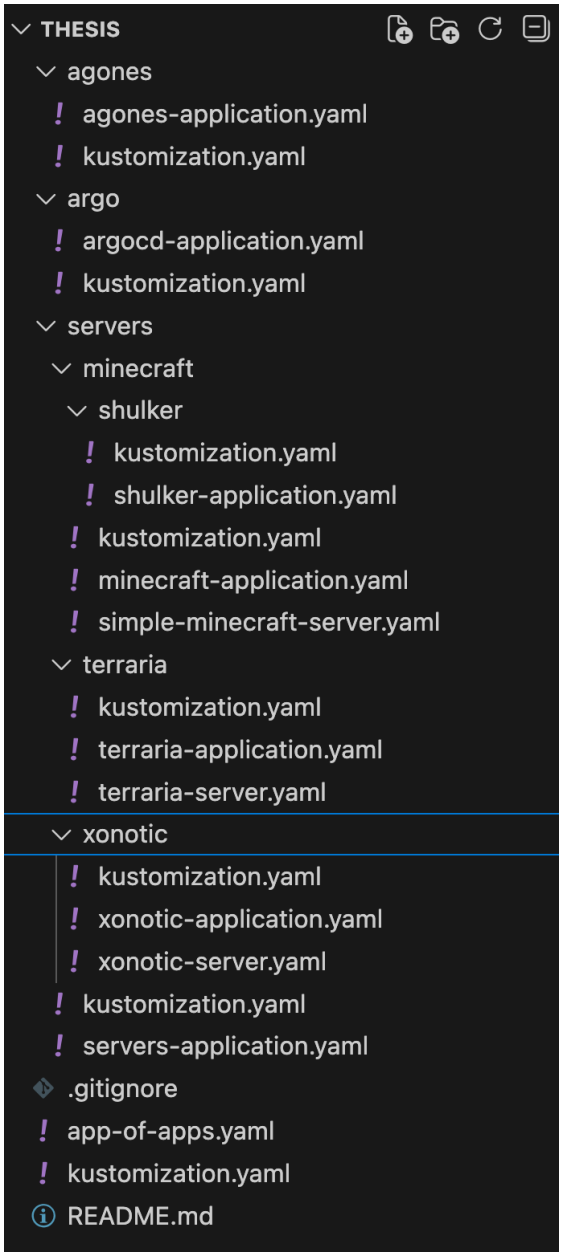
Projektet organiserades i en tydlig katalogstruktur för att underlätta GitOps-flödet och göra konfigurationen lätt att förstå och återanvända. Katalogen **/agones** innehåller ArgoCD-applikationen för Agones, som i sin tur installerar Agones via en Helm-chart.

Katalogen **/argo** innehåller motsvarande ArgoCD-applikation för ArgoCD själv, vilket gör att verktyget kan synkronisera och underhålla sin egen installation enligt GitOps-principerna.

Katalogen **/servers** är uppdelad i en mapp per spelserver, där varje mapp innehåller den kompletta konfigurationen för spelet, Fleet-resurser, Services samt en ArgoCD applikation som styr synkroniseringen av just den servern.

Varje mapp innehåller en `kustomization.yaml` fil, som används för att konfigurera och återanvända filerna med Kustomize, vilket gör det enkelt att lägga till nya spelservrar.

I detta projekt fungerar *app-of-apps* strukturen som grundnivå i GitOps-flödet. `App-of-apps.yaml` filen ligger i rotstrukturen av Git-repot och beskriver de alla tre mappar som ArgoCD ska synkronisera. Genom denna struktur blir ArgoCD ansvarigt för att installera och uppdatera alla komponenter automatiskt utifrån Git-repot, vilket eliminerar behovet av manuella installationer efter den initiala bootstrap-processen. Nya spelservrar kan läggas till genom att skapa en ny katalog och definiera en ArgoCD applikation för den.



Figur 3. Git-repostruktur.

4.2.1 Motivering till strukturen

Projektets katalogstruktur valdes för att göra hanteringen av spelservrar så tydlig och skalbar som möjligt. Genom att placera varje spel i en separat katalog blir det enkelt att konfigurera eller expandera miljön utan att gammal konfigurering påverkas. Varje katalog är definierad som en egen ArgoCD applikation, vilket innebär att ArgoCD kan synkronisera, övervaka och hantera varje komponent oberoende av de andra.

Kustomize bidrar ytterligare till denna modularitet genom att möjliggöra återanvändning av komponenter. Om ett spel vill köras på flera servar, till exempel två Minecraft servar med olik konfiguration, kan delarna återanvändas genom Kustomize i stället för att dupliceras. Katalogen *servers/* fungerar samtidigt som en logisk gruppering som gör det tydligt vilka resurser som hör till vilket spel, vilket förenklar både felsökning och framtida vidareutveckling.

Sammantaget ger denna struktur en ren, spårbar och skalbar grund som gör det lätt att bygga ut projektet och hålla ordning på alla resurser över tid.

4.3 ArgoCD-applikationerna

ArgoCD-applikationerna är en central del av projektets konfigurering och används för att tydligt separera de olika komponenterna och spel i klustret. Varje applikation är kopplad till en specifik del av Git-repot och definierar exakt vilka resurser som hör till en viss spelserver eller komponent. Detta gör det enkelt att arbeta modulärt, hålla komponenterna isolerade och säkerställa att varje del av klustret synkroniseras oberoende av de andra.

En ArgoCD-applikation består i huvudsak av två delar: **metadata** och **spec**. Metadata innehåller grundläggande information som resurstyp, applikationens namn och vilket Namespace den ska ligga i. I spec-blocket definieras repo-adressen, själva applikationens beteende, som vilken katalog i Git-repot ska användas, vilket Namespace resurserna ska installeras i, samt olika synkroniseringsregler.

I projektet har varje spelserver en egen applikation under `/servers`, vilket gör dem enkla att hantera som separata komponenter. ArgoCD-applikationen för Agones samt applikationen för ArgoCD:s egen installation hanteras på samma sätt, vilket ger hela projektet en enhetlig och skalbar struktur.

! argocd-application.yaml

```yaml
# Argo är GitOps verktyg för att synkronisera från denhär hit repon till kubernetes
apiVersion: argoproj.io/v1alpha1
kind: Application
metadata:
  name: argocd-app
  namespace: argocd
  finalizers:
    - resources-finalizer.argocd.argoproj.io
spec:
  project: default
  source:
    repoURL: https://argoproj.github.io/argo-helm
    chart: argo-cd
    targetRevision: 9.0.5
    helm:
      values: |
        server:
          extraArgs:
            - --insecure
  destination:
    server: https://kubernetes.default.svc
    namespace: argocd
  syncPolicy:
    automated:
      prune: true
      selfHeal: true
    syncOptions:
      - CreateNamespace=true
      - ServerSideApply=true
      - RespectIgnoreDifferences=true
    retry:
      limit: 5
      backoff:
        duration: 5s
        factor: 2
        maxDuration: 3m
  revisionHistoryLimit: 10
```

Figur 4. ArgoCD-application.yaml – Konfiguration på ArgoCD:s egna synkroniserings applikation.

4.4 Nätverksdesign

Nätverket spelade en viktig roll i projektet eftersom varje spelserver behövde exponeras via någon form av Kubernetes Service. De centrala komponenterna i miljön som ArgoCD och Agones, installerades med egna Services och fungerade direkt utan justeringar, vilket gjorde att fokus kunde ligga på spelservrarnas nätverksåtkomst.

Valet av portar var standardportar för spelen, och dessa behövde sedan exponeras på olika sätt beroende på protokoll. **TCP-trafik** fungerade utan problem med vanliga Kubernetes-services, medan **UDP-trafik** krävde extra konfigurering i den lokala datorn. För Xonotic behövdes en separat *socat*-forwarding för att tunnla UDP-trafik från klusteradressen till den lokala datorn, något som inte var nödvändigt för Minecraft eller Terraria.

Minecraft använde en LoadBalancer-service, eftersom detta ingick i standardkonfigurationen för Shulker Kubernetes. I ett vanligt molnkluster tilldelas varje LoadBalancer-service en egen extern IP-adress. Det innebär i praktiken att man normalt använder en LoadBalancer endast när det verkligen behövs, då varje service reserverar en unik, publik IP-adress. Xonotic och Terraria kunde däremot exponeras via NodePort-services, vilket är enklare och mer lämpligt för lokala installationer där spelservrarna endast behöver nås från värdmaskinen, och för extern access kan portforwarding användas på nätverksnivå (Kubernetes Documentation, u.å.-e). Vilket i projektets fall var en lokal fysisk router.

Sammanfattningsvis fungerade TCP-baserade spelservrar direkt via Kubernetes-services, medan UDP-baserade spel krävde extra manuell forwarding. Valet av LoadBalancer eller NodePort styrdes både av spelens standardkonfigurationer och av begränsningarna i den lokala utvecklingsmiljön.

4.5 Designbeslut och erfarenheter

Några centrala beslut som formade projektets implementation:

- **Valet av Rancher Desktop** gav en lättare att installera miljö än Minikube. Det kom också med en inbyggd GUI
- **GitOps med ArgoCD** gjorde konfigurationen stabil, reproducerbar och lätt att justera
- **Agones var helt nödvändigt** för ordentlig hantering av spelservrar
- **UDP-trafik i lokala kluster** var den största praktiska utmaningen

5 Jämförelse

I detta kapitel jämförs den Kubernetes-baserade lösningen i projektet med en mer traditionell spelserver-hosting där spelservrar installeras direkt på en virtuell eller fysisk server. Jämförelsen är inte baserad på egna mätningar av en traditionell miljö, utan på den praktiska erfarenheten från projektets Kubernetes-implementation samt normal praxis kring traditionell spelserver-hosting. Fokus ligger på fyra områden: installation och komplexitet, skalbarhet, uppdateringar samt reproducerbarhet och struktur.

De tre spelservrarna valdes eftersom det finns färdiga lösningar för att köra dem i Kubernetes med Agones, och eftersom jag sedan tidigare har personlig erfarenhet av att installera dem traditionellt på Linux eller Windows. Detta gjorde dem passliga för projektet.

5.1 Installation och komplexitet

Ur installationssynvinkel är en traditionell lösning oftast enklare. Att installera en spelserver direkt på en Linux- eller Windows-maskin innebär i regel en manuell installation med välkända verktyg som operativsystemets pakethanterare och serverspecifika binärer. Detta står i kontrast till Kubernetes-baserade installationer, som kräver ett kluster och flera extra komponenter. Traditionell hosting beskrivs därför som mer lättillgänglig och mindre komplex i små miljöer (Tieturi, 2025; CloudZenia, u.å).

Den Kubernetes-baserade lösningen i detta projekt har en högre initial komplexitet. Först behöver ett kluster sättas upp, i detta fall via Rancher Desktop, därefter installeras ArgoCD, Agones och spelservrarna. Med det krävs en förståelse för YAML, GitOps och Kubernetes. Erfarenheten från projektet är att tröskeln in är betydligt högre, men att komplexiteten blir mer motiverad ju fler komponenter som ska hanteras. För ett enskilt spel och en enskild server är den traditionella modellen sannolikt mer tillgänglig.

5.2 Skalbarhet

När grundmiljön är på plats är skalbarheten en tydlig styrka med Kubernetes. I projektet räcker det ofta med att kopiera en katalog i Git-repot, göra mindre justeringar och skapa

en ny Kustomize-konfiguration för att få flera spelservrar. Horisontell skalning hanteras genom Fleets och deklarativa inställningar för antal instanser.

I en traditionell miljö är horisontell skalning mer manuell. Att lägga till fler servrar innebär typiskt nya virtuella maskiner och konfiguration (CloudZenia, u.å). Vertikal skalning; att ge en server mer CPU och minne, är däremot ungefär lika enkel i båda modellerna. I Kubernetes fall att ändra YAML-konfiguration, och i traditionella fall att ändra konfiguration som kommer med spelets binär. Om man ändå behöver en maskin som orkar köra tre spelservrar samtidigt, blir själva overheaden från Kubernetes relativt liten, eftersom majoriteten av hårdvaran används av spelen.

Sammanfattningsvis framstår Kubernetes-lösningen som mer skalbar när antalet spelservrar växer, medan med enstaka servrar kan traditionell hosting vara lättare.

5.3 Uppdateringar

Uppdateringar är ett område där Kubernetes och GitOps tydligt förenklar arbetet. I projektet räcker det med att ändra versionen i en YAML-fil i Git-repot för att uppgradera en server, varefter ArgoCD ser till att klustret uppdateras. En nedgradering blir lika enkel. Man kan gå tillbaka till en tidigare commit eller justera versionen manuellt och låta ArgoCD rulla ut ändringen.

I en traditionell miljö sker uppdateringar ofta genom manuella kommandon, skript eller direktkonfiguration på servern, vilket gör processen långsammare. Det är fullt möjligt att skapa strukturerade processer även där, men det kräver en del arbete. Nedgraderingar kan särskilt vara besvärliga om installationer skrivs över (CloudZenia, u.å; Tieturi, 2025).

Projektet visar att Kubernetes gör hantering mera reproducerbart och mindre beroende på manuella steg. Detta blir viktigare ju fler servrar och spel som ska administreras. Nackdelen här med Kubernetes är kravet för färdiga containerimages. I fallet av skräddarsydda Kubernetes lösningar, där man hamnar uppdatera eller göra nya containerimages, är

mängden arbete samma eller möjligt mera än i traditionella installationer. Här blir valet oftast fast på om det finns färdiga containerimages för spelet man vill hosta.

5.4 Reproducerbarhet och struktur

Reproducerbarhet och struktur är där Kubernetes- och GitOps-ansatsen är som tydligast överlägsen. I detta projekt finns all relevant konfiguration samlad i Git-repot. Klustrets applikationer, Agones-resurser och spelserverdefinitioner. Om allt skulle behöva installeras om från början räcker det i princip med att sätta upp ett nytt kluster, installera ArgoCD och låta ArgoCD synkronisera hela miljön från repot. Konfigurationen fungerar samtidigt som dokumentation, den beskriver exakt hur systemet är tänkt att se ut.

I en traditionell miljö krävs oftast separata installationsguider, skript eller manuella rutiner för att uppnå samma grad av reproducerbarhet som i en GitOps-miljö, vilket gör att konfiguration lätt sprids ut i diverse dokument (CloudZenia, u.å; Tieturi, 2025). Detta ökar risken för så kallade “snowflake servers”, där manuella ändringar gör att servrar gradvis blir unika och svåra att återskapa. Från projektets resultat visar Kubernetes som klart mer reproducerbar, ifall användaren kan läsa och förstå YAML-filerna och Git-strukturen.

5.5 Sammanfattning av jämförelsen

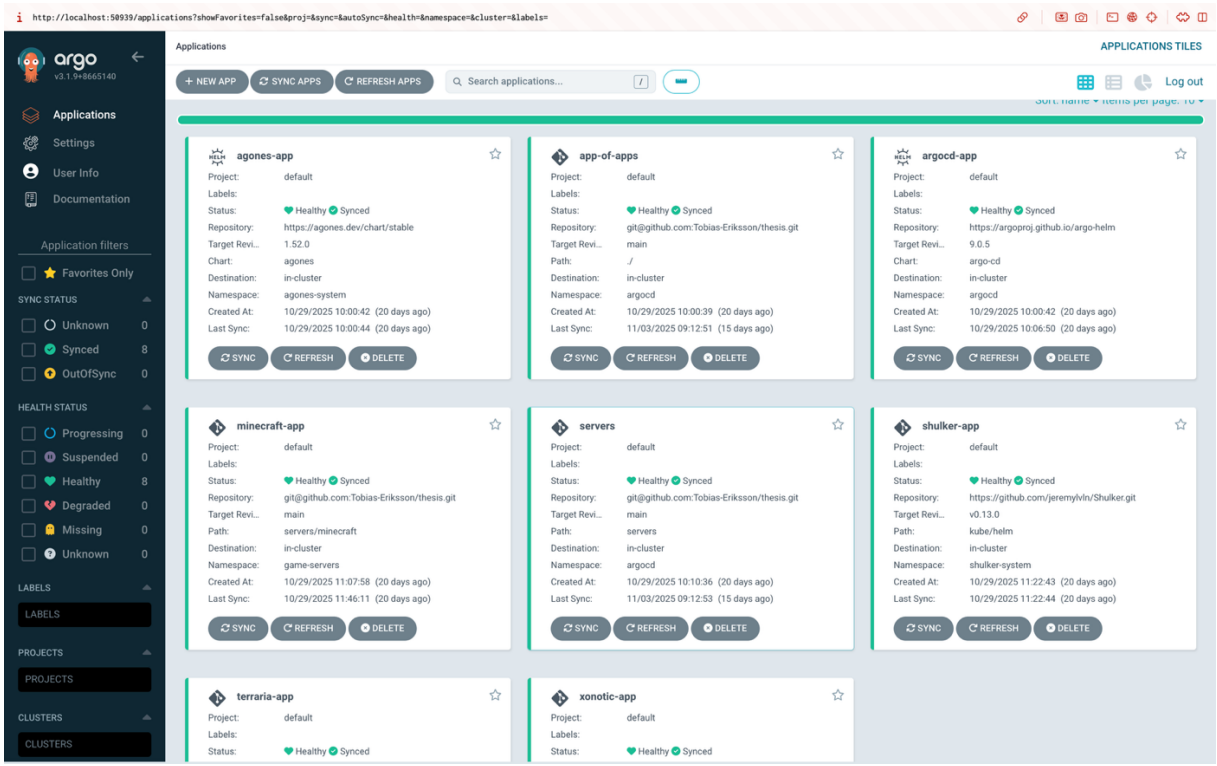
Resultaten visar att traditionell spelserver-hosting har en fördel av mindre initial komplexitet, särskilt då man jämför små traditionella servrar eller mindre testmiljöer. Kubernetes-baserade lösningen är däremot starkare när det gäller skalbarhet, hantering, uppdateringar samt reproducerbarhet och struktur. För små hobbyprojekt kan båda sätten vara rimliga. Om målet är att hantera flera spelservrar på ett reproducerbart sätt med tydlig dokumentation, kan Kubernetes och GitOps vara ett mera långsiktigt hållbart alternativ, medan i fallet av enstaka servar eller testmiljöer, kan traditionell hosting vara rimligare alternativet.

6 Resultat

Detta kapitel presenterar de huvudsakliga resultaten av projektets implementation. Resultaten baseras på den genomförda installationen av Kubernetes, ArgoCD och Agones, samt konfigurationen och körningen av de tre spelservrarna Minecraft, Xonotic och Terraria. Syftet är att visa vad som faktiskt fungerade i praktiken och vilka tekniska problem kom fram under projektets gång.

6.1 Fungerande Kubernetes- och GitOps-miljö

Projektet resulterade i en fullt fungerande lokal Kubernetes-miljö med Rancher Desktop, där både ArgoCD och Agones installerades och drevs utan kritiska problem. ArgoCD visade applikationerna som ”Synced + Healthy”, vilket visade att GitOps-flödet var fungerande och klustret motsvarade den deklarerade konfigurationen. Konfigurationer för applikationer kunde ändras direkt via Git och rullades därefter ut automatiskt till klustret.



Figur 5. ArgoCD visar applikationerna som ”Synced + Healthy”.

6.2 Spelserverdrift med Agones

De tre spelservrar kunde köras i Kubernetes med hjälp av Agones. Resultaten för de olika spelen var följande.

6.2.1 Minecraft

Minecraft-servern, som implementerades via Shulker Kubernetes, fungerade efter att Fleet-, Cluster- och Proxy-resurserna synkroniserats via ArgoCD. Servern startade stabilt och kunde anslutas till. Spelet delades upp till lobby- och en public-pod.

6.2.2 Xonotic

Xonotic, som använder en officiell exempelimage från Agones, fungerade utan problem efter drift. Den enda observerade utmaningen var att Xonotic använder UDP-trafik, vilket krävde manuell *socat*-forwarding. När detta var löst gick det att ansluta till servern som förväntat.

6.2.3 Terraria

Terraria visade sig vara det enklaste spelet att driftsätta. Med en Agones Fleet-resurs och en enkel Kubernetes Service kunde servern startas upp utan justeringar. Då spelet använder TCP-trafik fungerade anslutning utan komplikationer.

Figur 6. Spel- samt shulker-pods snurrar på klustret.

6.3 GitOps-reproducerbarhet

Ett av projektets tydligaste resultat är att hela systemet kan återskapas från Git-repot. Genom att installera klustret, ArgoCD och genom att synkronisera Git-repot, kunde alla komponenter återskapas automatiskt. Agones-installationen, Fleet-resurserna och spelservrarna. Detta innebar att miljön kunde återställas vid behov och att konfigurationerna fungerade som dokumentation.

6.4 Tekniska observationer

Under projektets gång framkom flera tekniska observationer som påverkade arbetet och gav en tydligare bild av hur lösningen fungerar i praktiken. Kubernetes-miljön i Rancher Desktop visade sig vara stabil även med flera spelservrar i gång samtidigt, vilket visade att plattformen klarar av parallell belastning i en lokal miljö. GitOps-modellen förenklade hanteringen av uppdateringar betydligt. Version byten kunde göras genom små ändringar i YAML-filerna, och ArgoCD såg till att dessa rullades ut konsekvent till klustret.

Det blev också tydligt att nätverkstrafik skiljer sig åt beroende på spelens protokoll. TCP fungerade i stort sett utan justeringar, medan UDP-beroende applikationer, såsom Xonotic, krävde extra portforwarding i den lokala miljön. När det gäller Agones, hanterade systemet de deklarerade Fleet-resurserna på ett förutsägbart sätt, och skapade automatiskt GameServer- och stödresurser enligt deklarerade konfigurationen. Spelservrarna varierade i komplexitet. Minecraft hade den mest komplexa struktur, pga. Användning av Shulker-projektet. Terraria var däremot den enklaste servern att driftsätta, då den behövde få komponenter och fungerade nästan direkt med en standard Fleet och Service.

6.5 Sammanfattning av resultaten

Sammanfattningsvis visar resultaten att:

- En fullständig och fungerande GitOps-baserad Kubernetes-miljö går att åstadkomma även i liten skala.
- Agones är väl lämpat för att köra och hantera spelservrar på ett strukturerat sätt.

- Alla tre spelservrar kunde köras parallellt och hanteras via Agones utan större problem.
- Miljön är reproducerbar och kräver relativt lite manuellt arbete efter installation.

Resultaten är grunden för diskussionen i nästa kapitel, där lösningens styrkor, begränsningar och möjliga förbättringar analyseras.

7 Diskussion

Detta kapitel diskuterar projektets resultat, de tekniska valen och de erfarenheter som gjordes under implementationen. Diskussionen fokuserar på lösningens styrkor, begränsningar och vad projektet säger om användningen av Kubernetes och GitOps i mindre spelsammanhang.

7.1 Bedömning av lösningen

Projektets huvudmål var att visa ett praktiskt sätt att köra spelservrar i Kubernetes och att undersöka om DevOps- och GitOps-praktiker är användbara även i mindre eller hobbybaserade projekt. Resultaten visar tydligt att detta är fullt möjligt. Med ArgoCD och Agones kunde flera spelservrar hanteras deklarativt och reproduceras på ett strukturerat sätt. Samtidigt framkom det att den initiala komplexiteten är betydligt högre jämfört med traditionell hosting. I detta projekt krävdes förståelse på Kubernetes, mikroarkitektur och diverse resurstyper. Medan GitOps-modellen bygger på det med ännu mera krav. För användare utan tidigare erfarenhet kan detta vara ett hinder.

Trots detta visar projektet att när den första tröskeln är passerad blir arbetet både mer organiserat och enklare att hantera i längden. Detta visar att Kubernetes-baserad hosting passar särskilt bra när man har flera spelservrar eller vill arbeta systematiskt med uppdateringar och konfigurationer.

7.2 Erfarenheter från spelserverdriften

De tre spelservrarna gav olika erfarenheter. Minecraft, med sin mer avancerade struktur via Shulker Kubernetes, visade att även komplexa spel kan anpassas bra till Kubernetes om man använder färdiga opensource-lösningar. Terraria var på andra sidan så enkel att den nästan inte krävde något annat än en Fleet och en Service. Xonotic visade tydligt skillnaden mellan TCP- och UDP-baserade spel i lokala kluster. Här blev nätverk en verklig praktisk utmaning.

En viktig observation är att spelen skiljer sig på hur enkelt containerisering för dem är. Vissa spel är byggda för containerdrift, medan andra kräver mera skräddarsydda lösningar. Det innebär att Kubernetes och Agones inte automatiskt passar alla spel.

7.3 Styrkor och begränsningar i GitOps-modellen

GitOps visade sig vara en tydlig styrka i projektet. Att uppdatera, rulla tillbaka och reproducera systemet från Git-repot var enkelt och pålitligt. Detta gav projektet en professionell struktur som är lätt att underhålla och förstå.

Begränsningen är att GitOps är beroende av att den som använder systemet förstår filerna och flödet, och att små misstag i YAML kan orsaka problem. I små projekt kan GitOps kännas som ett större verktyg än vad som behövs, men för projekt av denna typ, där flera komponenter och servrar ska hanteras, blev det användbart och även nödvändigt.

7.4 Reflektioner kring projektets metoder och avgränsningar

En avgränsning i projektet var att den traditionella hostingmodellen inte implementerades praktiskt, utan jämförelsen baserades på allmän praxis och erfarenheter från Kubernetes-delen. Detta är lämpligt med tanke på projektets syfte, som var att beskriva och analysera Kubernetes- och GitOps-lösningen, inte att genomföra två fullständiga implementationer.

8 Slutsats

Syftet med detta arbete var att utveckla och dokumentera en praktisk lösning för game hosting i Kubernetes och att undersöka hur DevOps- och GitOps-praktiker kan tillämpas i ett mindre projekt. Resultaten visar att lösningarna är fullt användbara och att Kubernetes, ArgoCD och Agones tillsammans bildar en kraftfull plattform för både hantering och reproduktion av spelservrar.

Projektet demonstrerar att GitOps ger en tydlig struktur som underlättar uppdateringar, skalning och återställning. Agones visade sig effektivt för att hantera livscykeln för spelservrar och gjorde det möjligt att köra både komplexa och enkla spel parallellt i klustret. Samtidigt blev det tydligt att Kubernetes innebär en hög initial komplexitet och att vissa spel fortfarande kräver specialanpassningar för att fungera bra i containrar.

Den slutliga sammanfattningen är att Kubernetes och GitOps är mest värdefullt när man vill hantera flera spelservrar eller skapa en stabil och reproducerbar infrastruktur. För små, enskilda spelservrar kan en traditionell hostinglösning vara enklare att komma i gång med, men saknar de fördelar som GitOps ger i form av struktur, spårbarhet och automatisering.

Detta arbete visar att avancerade DevOps-principer inte bara är användbara i stora företagsmiljöer, utan även kan ge tydliga fördelar i mindre, hobbyinriktade projekt när kraven på struktur, skalbarhet och reproducerbarhet är viktiga.

Källor

CNCF. (u.å). *CNCF Projekt webbsida för Kubernetes.* https://www.cncf.io/projects/kubernetes/

Kubernetes. (2024). *10 years of Kubernetes.* https://kubernetes.io/blog/2024/06/06/10-years-of-kubernetes/

Kubernetes Documentation. (u.å.-a). *Declarative Management of Kubernetes Objects Using Configuration Files.* https://kubernetes.io/docs/tasks/manage-kubernetes-objects/declarative-config/

Kubernetes Documentation. (u.å.-b). *Control Plane Components.* https://kubernetes.io/docs/concepts/overview/components/#control-plane-components

Kubernetes Documentation. (u.å.-c). *Nodes.* https://kubernetes.io/docs/concepts/architecture/nodes/

Kubernetes Documentation. (u.å.-d). *Cluster Architecture.* https://kubernetes.io/docs/concepts/architecture/

CNCF. (2021). *GitOps 101: What's it all about?* https://www.cncf.io/blog/2021/09/28/gitops-101-whats-it-all-about/

Red Hat. (2025). *What is Infrastructure as Code (IAC)?* https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac

ArgoCD Documentation. (u.å). *Argo CD – Declarative GitOps CD for Kubernetes*. https://argo-cd.readthedocs.io/en/stable/

Agones Documentation. (u.å-a). *Agones Kubernetes API*. https://agones.dev/site/docs/reference/agones_crd_api_reference/

Agones Documentation. (u.å-b). *Fleet Specification*. https://agones.dev/site/docs/reference/fleet/

googleforgames/agones. (2018). *GitHub Issue – GameServer Fleets #70*. https://github.com/googleforgames/agones/issues/70

Kubernetes Documentation. (u.å.-e). *Service type.* https://kubernetes.io/docs/concepts/services-networking/service/#publishing-services-service-types

CloudZenia. (u.å). *Kubernetes vs. Traditional Cloud Hosting: Key Differences Explained.* https://cloudzenia.com/blog/kubernetes-vs-traditional-cloud-hosting-what-is-the-difference/

Tieturi. (2025). *The Benefits of Kubernetes as a Platform for Modern Software Development.* https://www.tieturi.fi/en/blogi/the-benefits-of-kubernetes-as-a-platform-for-modern-software-development/