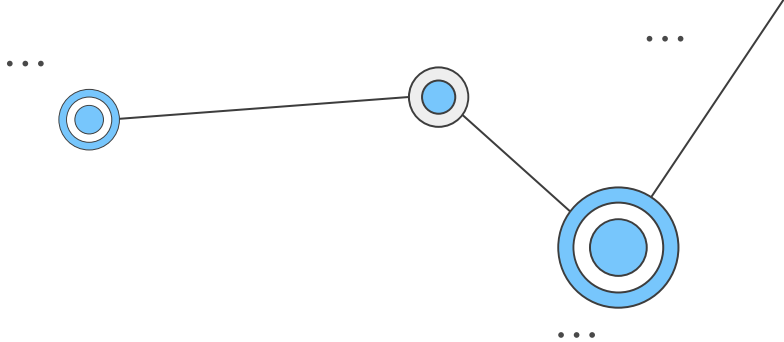


Building a scalable ecosystem for high-loaded multiplayer game

Dmitrii Ivashchenko | MY.GAMES

About

Dmitrii Ivashchenko

Lead Software Engineer

Projects

Overview

CI/CD Organization

How to get a large team
up and running

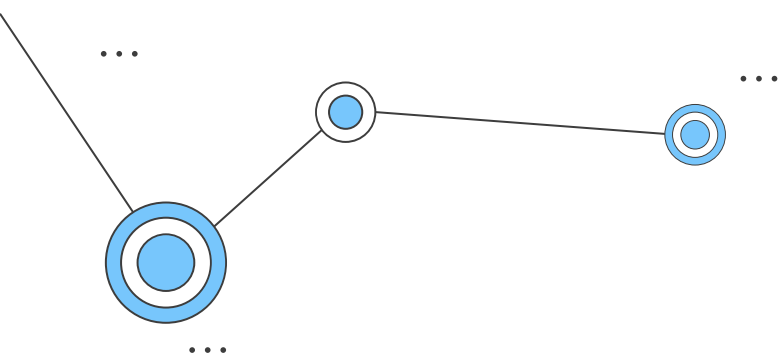
Backend Infrastructure

How to prepare the
backend for scaling up

Blue/Green Deployment

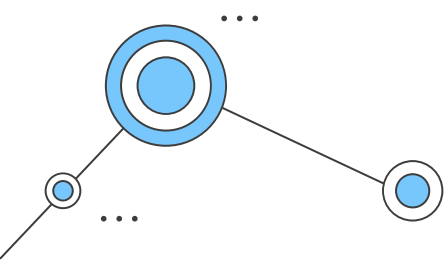
How to release new versions
without stopping for technical work

CI/CD

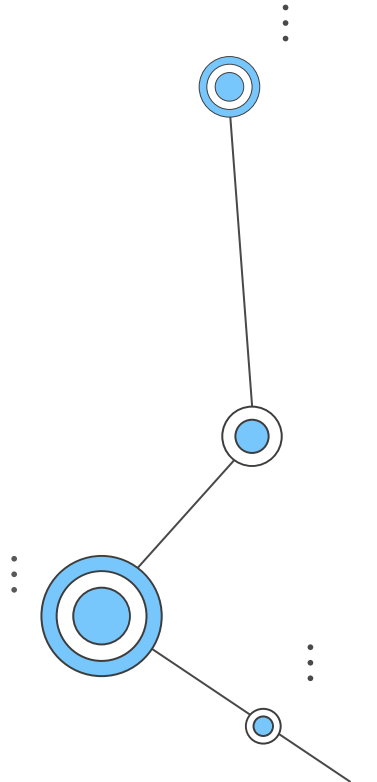


Goals and Principles

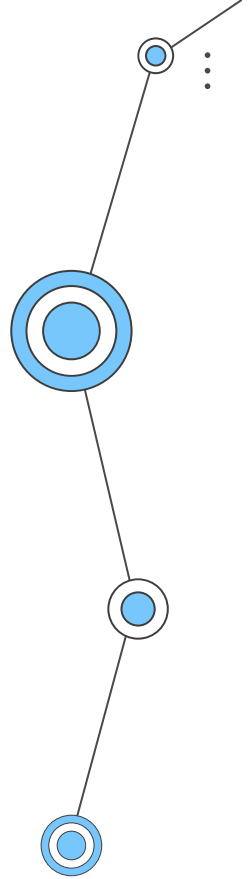
- Using containers for maximum environment reproducibility for CI and staging.
- Using an emulator farm for testing.
- Simplifying the deployment of test and production servers to the "click of a button" level.
- Do the maximum possible checks at the merge request stage.
- Infrastructure as Code paradigm.



Infrastructure as Code

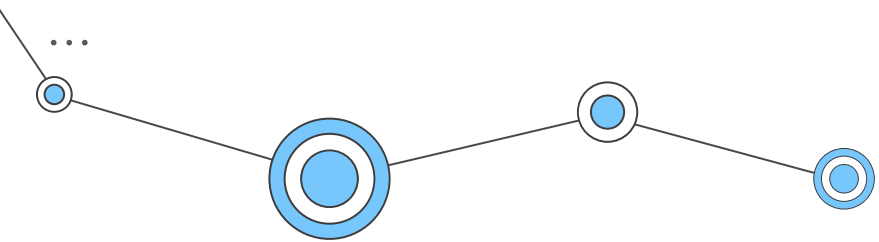


- Versioning
- Automation
- Repeatability and Consistency
- Documentation
- Scalability and Changes
- Collaboration and Responsiveness

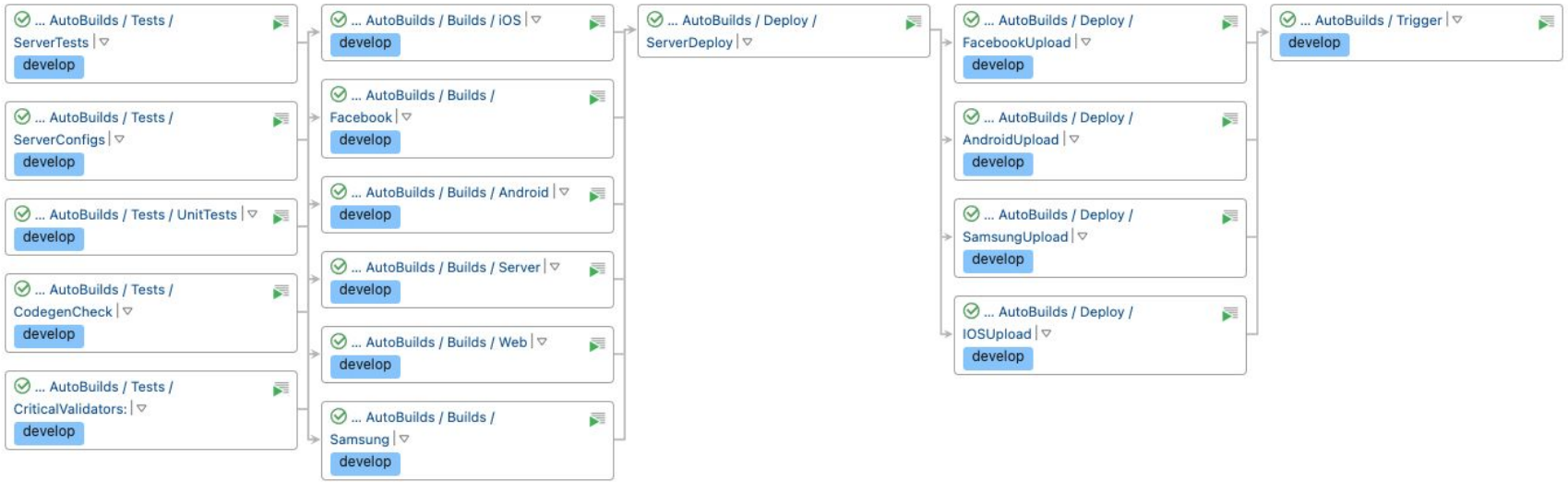


TeamCity and GitLab

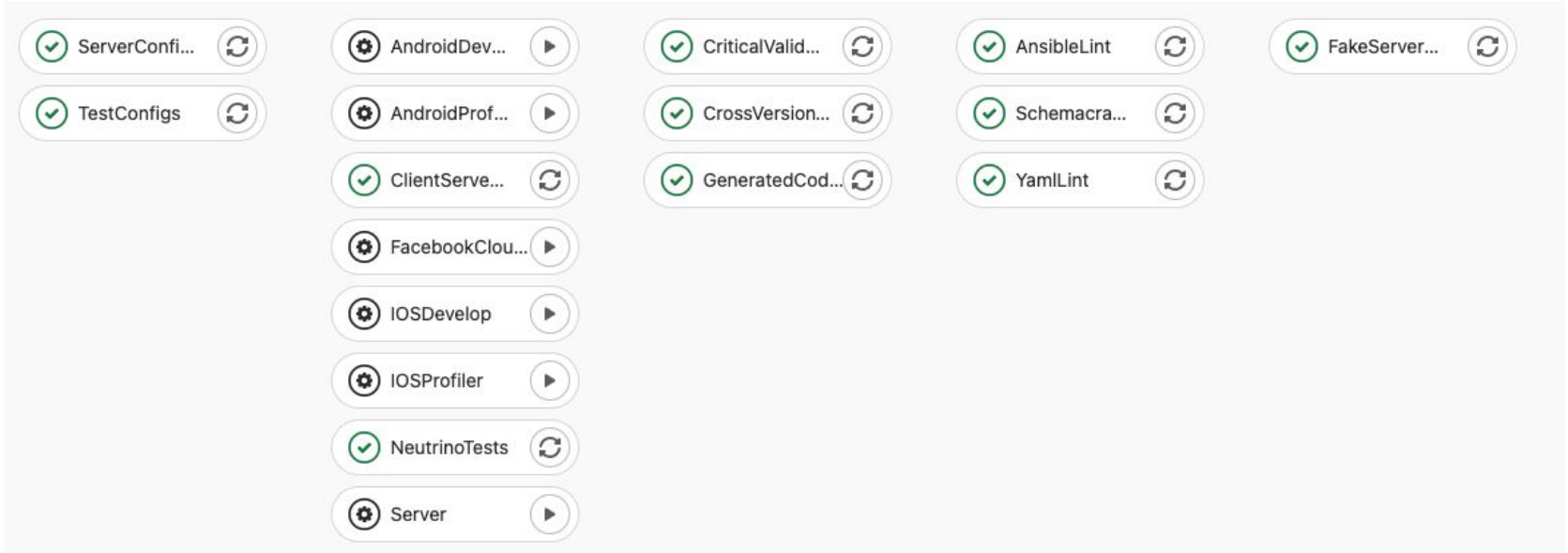
- Teamcity allows storing all configuration in Kotlin DSL.
- GitLab allows doing this with YAML.
- In GitLab, it's convenient to perform checks after pushing.



TeamCity


... AutoBuilds / Tests / ServerTests
develop
... AutoBuilds / Tests / ServerConfigs
develop
... AutoBuilds / Tests / UnitTests
develop
... AutoBuilds / Tests / CodegenCheck
develop
... AutoBuilds / Tests / CriticalValidators:
develop
... AutoBuilds / Builds / iOS
develop
... AutoBuilds / Builds / Facebook
develop
... AutoBuilds / Builds / Android
develop
... AutoBuilds / Builds / Server
develop
... AutoBuilds / Builds / Web
develop
... AutoBuilds / Builds / Samsung
develop
... AutoBuilds / Deploy / ServerDeploy
... AutoBuilds / Deploy / FacebookUpload
develop
... AutoBuilds / Deploy / AndroidUpload
develop
... AutoBuilds / Deploy / SamsungUpload
develop
... AutoBuilds / Deploy / IOSUpload
develop
... AutoBuilds / Trigger
develop

GitLab


ServerConfi...
TestConfigs
AndroidDev...
AndroidProf...
ClientServe...
FacebookClou...
IOSDevelop
IOSProfiler
NeutrinoTests
Server
CriticalValid...
CrossVersion...
GeneratedCod...
AnsibleLint
Schemacra...
YamlLint
FakeServer...

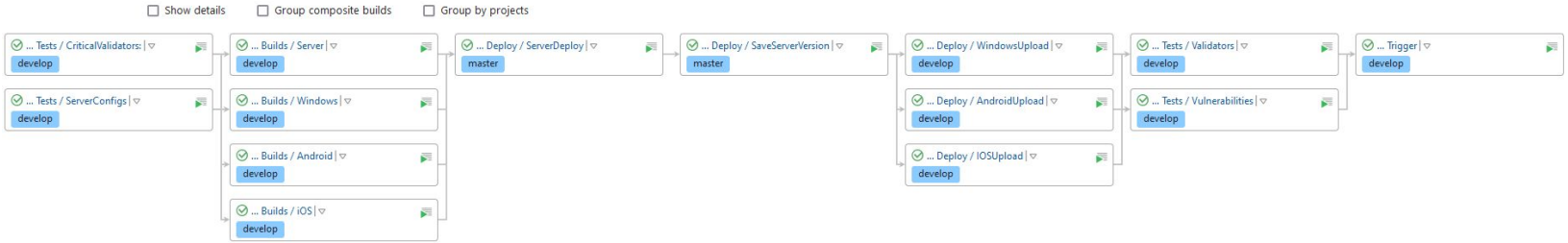
AutoBuilds

- **Automatic Builds**: Builds triggered for specific branches (Develop and Stable).
- **Exclusions**: Excludes builds for feature branches or task-specific testing.
- **Server and Client Builds**: Builds server and clients for three main platforms (PC, Android, iOS) after validation and server configuration.
- **Server Launch**: Launches a server named "Develop" after successful builds.
- **App Center Upload**: Uploads client versions to the App Center for later downloads to devices.
- **Regular Updates**: Ensures a fresh client-server pair with the latest changes every N hours.

Server

Steps to build a server:

1. Assemble server configurations.
2. Build the server based on these configurations.
3. Deploy the server.



OnDemand Servers

Isolated Testing Environment

OnDemand servers offer a separate, isolated environment ideal for testing new features, bug fixes, or running experiments without impacting the main development flow.

Automated Server Setup

The build process in Teamcity creates server configurations, the server itself, and a cloud-based virtual machine that automatically launches the server.

User-Specific Access

Each Teamcity user is assigned a unique name for their virtual machine, allowing for individualized testing environments.

Scheduled Deletion

To manage resources, servers created in this manner are automatically deleted twice a week.

Merge Request Workflow

Branch Creation and Committing

Developers create a new branch from the "develop" HEAD, make task or bug-specific commits, and push these to the branch.

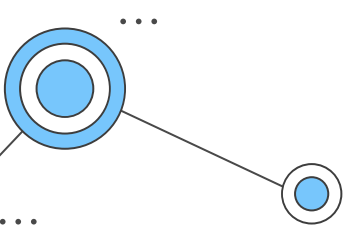
Merge Request and Reviews

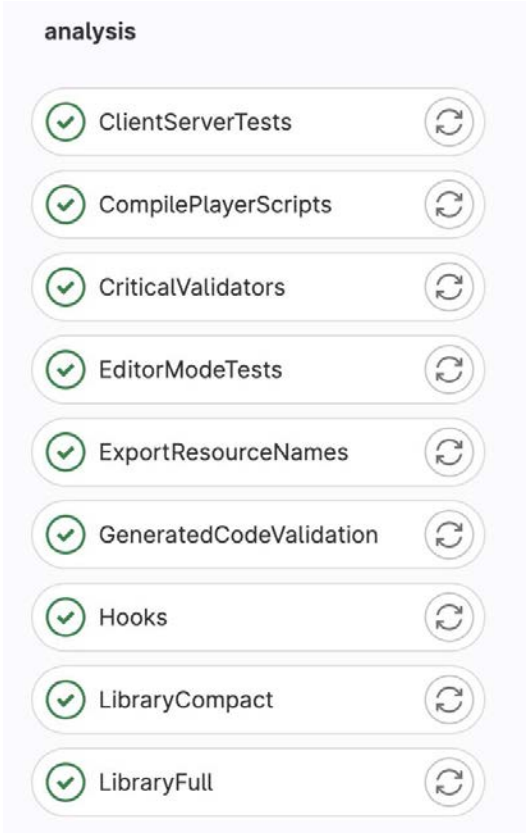
An MR is created for the new branch, triggering automated tests and optional manual reviews.

Closing Outdated MRs

Irrelevant or outdated Merge Requests should be closed to avoid clogging the MR list and causing confusion.

Validation System





Validation System

Connecting to the Project

The Validator System is stored as an AssetValidation package and can be added to the Unity project like any other package.

Running and Configuring Validators

Validators can be run individually or in groups via a special window accessed from the Unity top panel.

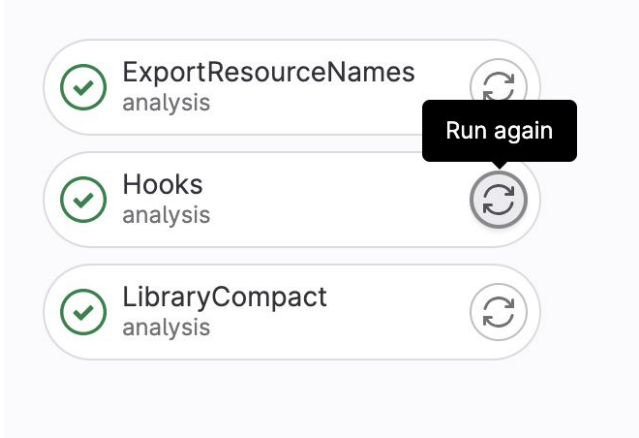
Custom Selectors and Validator Types

Validators come in four types that determine their behavior and the types of assets they validate.

Field Change Validators

Special validators can be created to track and validate changes in config fields that cannot be automatically verified.

Git Hooks


ExportResourceNames
analysis
Run again
Hooks
analysis
LibraryCompact
analysis

Client Hooks

Installer Component

A binary for installing and automatically updating Git hooks in the local repository copy.

Commit-Msg Hook

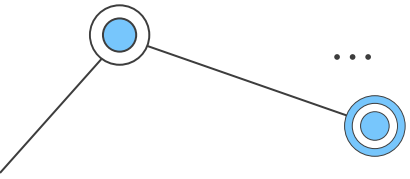
This hook validates the commit message according to predefined rules. It serves to enforce best practices for commit messages.

Pre-Commit

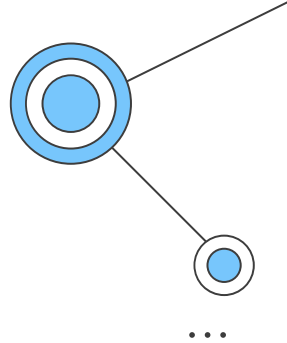
This hook is responsible for the majority of file checks before a commit is finalized. It uses specific rules to validate the files that are about to be committed.

Post-Commit

These hooks are executed after a commit. Mainly, they are used for various notifications.



Server Hooks



ProtectedBranch

Restricts pushes to specified branches, making them read-only for users who try to push changes.

RebaseRequired

Requires that a rebase is performed before allowing a push to proceed, ensuring that the branch is up-to-date with the main repository.

NewBranchName

Validates the name of new branches.

MessageContent

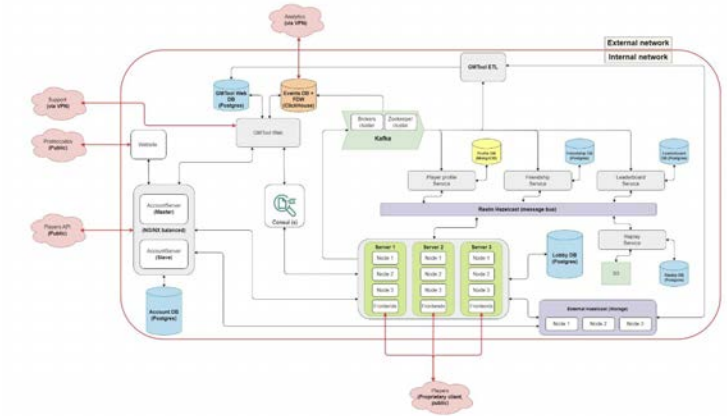
Enforces specific formatting rules for commit messages by utilizing regular expressions.

Backend Infrastructure

Platform Architecture

Eclipse Vert.x

Serves as the messaging system and provides clustered storage for runtime data.



Ansible

Responsible for configuring server applications, automating the setup and maintenance processes.

Hazelcast

Functions as an in-memory data grid and serves as the foundation for Vert.x.

Apache Kafka

Acts as a log data broker, handling the flow and storage of log data.

PostgreSQL

Utilized for storing persistent data through various methods and formats.

Main Platform Components

Account Server

Responsible for user authentication and holds information about all connected game servers.

Game Server

Acts as the main repository for game mechanics, logic, and data.

Gametool WEB

Serves as an administration tool for both players and servers, facilitating easier management.

Gametool ETL

Extracts game logs from Apache Kafka and loads them into the Gametool database for further analysis.

Account Server Components

Game-Servers Configuration Component

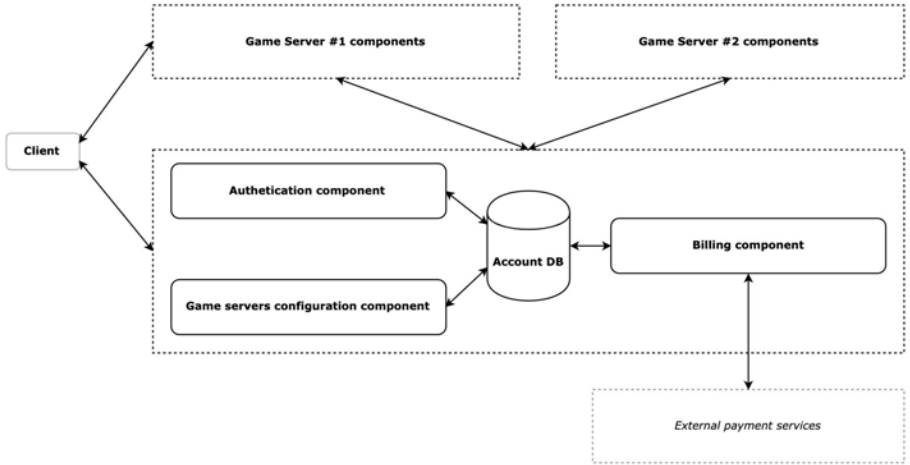
Manages communication with game servers, announces maintenance, and other administrative tasks.

Authentication Component

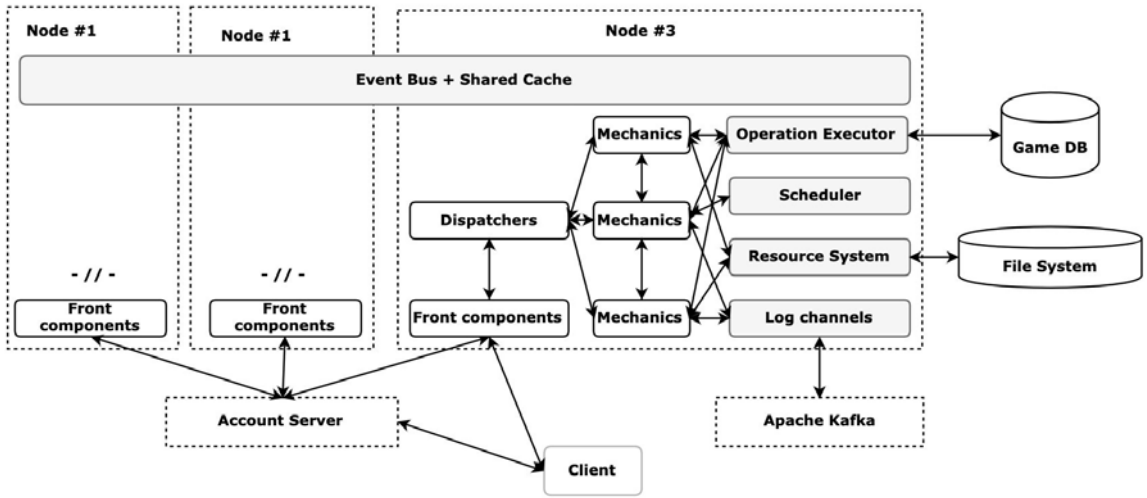
Responsible for user authentication and distribution to game servers and front-components.

Billing Component

Processes in-game purchases.



Game Server Architecture



- Cluster Nodes
- Front Component
- Dispatcher
- Scheduler
- DB Operation Executor
- Resource System
- Log System
- Mechanic Components

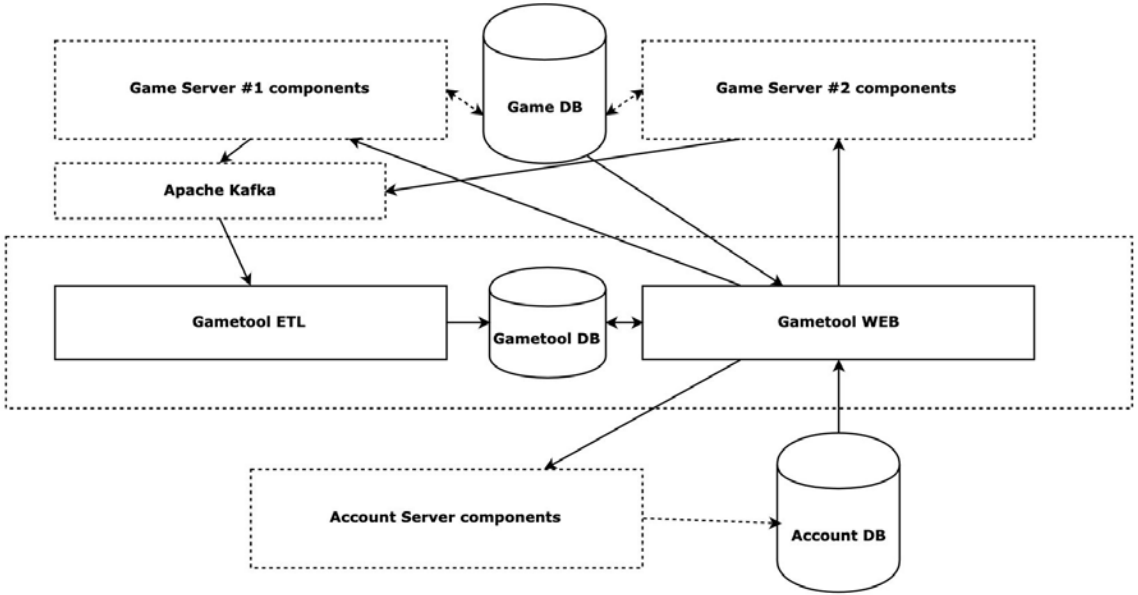
GameTool Architecture

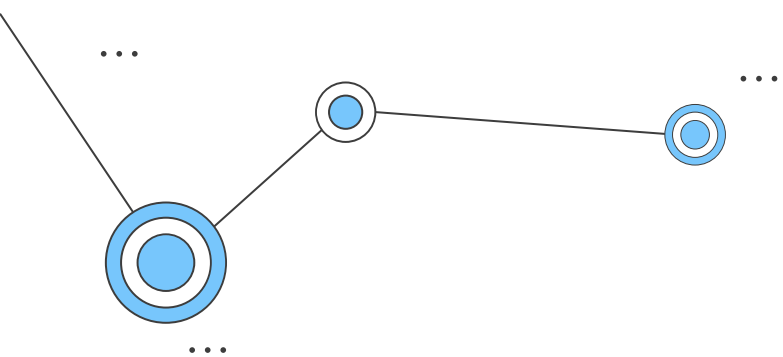
GameTool ETL

Extracts game logs from Apache Kafka, processes the data, and stores the transformed data in its own database.

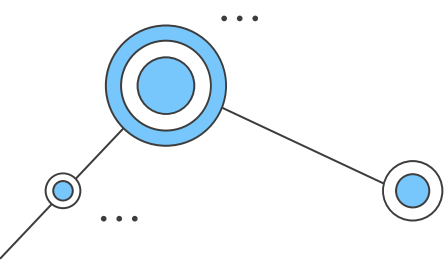
GameTool WEB

Serves as a server administration tool, allowing access to player information and logs.



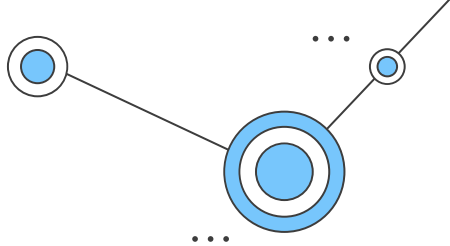


Mechanics Services

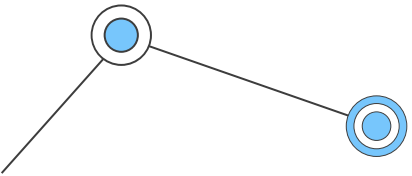


- **Proton**: Used for PvP and cooperative gameplay.
- **Leaderboards**: A system for storing and processing player rankings.
- **Friends**: Manages the list of friends and referral information.
- **Player Profile**: Provides a detailed card of the player's information.
- **Replay**: Designed to store gameplay replays.
- **Mail**: Aimed at storing and processing in-game mail messages.
- **Chat**: Processing of messages, members, and settings for all chats.
- **Match Making**: Designed for finding opponents and teammates.
- **Clans**: Manage clans and clan activities.
- **Push Notifications**: Sending notifications to devices.

Photon Cloud



Our current architecture uses Photon Cloud to coordinate real-time multiplayer gaming and other functionalities, offering low-latency data centers worldwide.



Data Storage and Messaging

PostgreSQL
Primary Data Storage

Apache Kafka
Message Brokering

Hazelcast
In-Memory Database

Vert.x
Reactive Application Framework

Vert.X

Vert.x Advantages

Supports multiple programming languages and operates on a reactor pattern.

Vert.x Challenges

Can lead to complicated code if the programming language used is not fully supported by the framework.

Quasar as an Alternative

Considered as a potential alternative to Vert.x but was not actively maintained as of 2017.

Handling Transactional Operations

Transactions

Created to allow linear operation within message processing and covers most use-cases.

Testing Findings

Discovered a lock queuing issue in Vert.x during testing.

Vert.x Updates

Developers have addressed the lock queuing issue reported during testing.

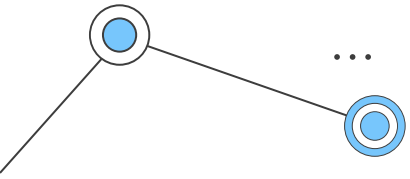
Prometheus and Grafana

Monitoring with Prometheus

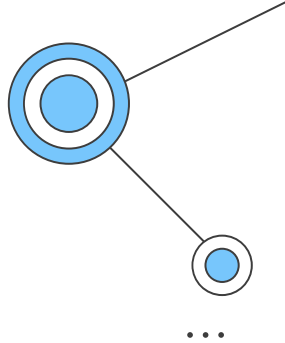
Used to collect performance metrics.

Visualization with Grafana

Utilized to visualize the metrics gathered by Prometheus.



Game Cluster Architecture



Cluster Composition

Collection of machines running instances of Vert.x and Hazelcast.

Node Functionality

Each node runs various game mechanics.

Vert.x 'Verticles'

Encapsulate different tasks such as game model loading or arcade tasks.

Admin Interface

Used for comprehensive management of the entire setup.

Scaling Strategy

Current Capacity

Hardware can comfortably support up to 150,000 CCU.

CPU Limitations

Additional servers can be added to the cluster if CPU limits are reached.

PostgreSQL Bottleneck

Identified as a potential first bottleneck in scalability.

Hazelcast Solution

Deferred synchronization with Hazelcast can help alleviate PostgreSQL scalability issues.

Blue/Green Deployment

Reasons for Adopting Blue-Green Deployment Strategy

Expense Consideration

Architectural and manufacturing costs associated with Blue-Green Deployment (BGD) need to be weighed against benefits.

Downtime Costs

Even 1 minute of downtime is expensive, making BGD beneficial for minimizing or eliminating downtime.

Mobile Game Publishing

New versions require store approval, which takes time.

Blue-Green Deployment in Practice

- **Components Involved**: A client and two servers (Alpha and Beta) are the main elements of the setup.
- **Traffic Switching**: The aim is to switch traffic from Alpha to Beta seamlessly, without player interruptions.
- **Roles of Parties**:
  - **Game Server**: Runs the game and interacts with the client.
  - **Client**: Connects to the game server for gameplay.
  - **Special Account Server**: Provides the client with the address of the game server to connect to.
- **Account Server's Knowledge**: Knows the game server address and its status (live/stopped), which is meta-information unrelated to the actual running status of the game server.

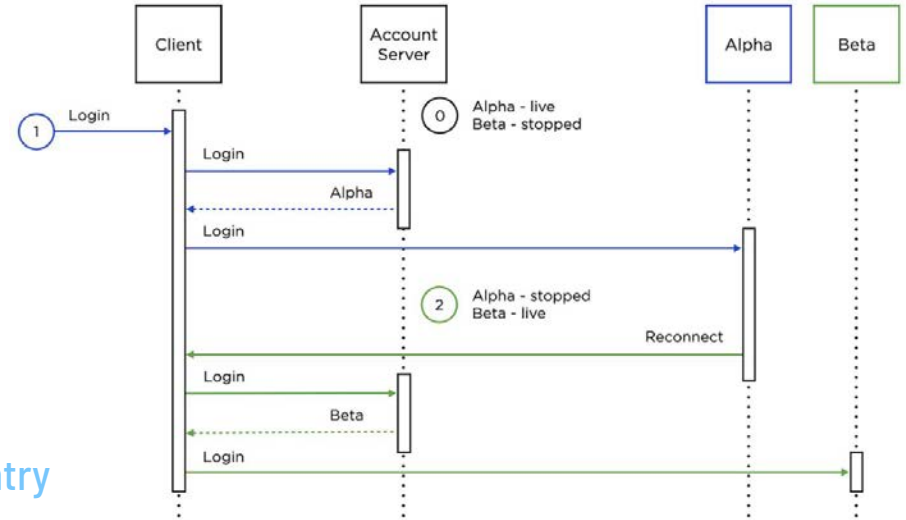
Zero Downtime Server Update

Initial Status

Alpha server is live,
Beta server is stopped.

Player Entry

When a player enters the game, the client contacts the account server, which provides the Alpha server's address for the client to connect.



Status Switch

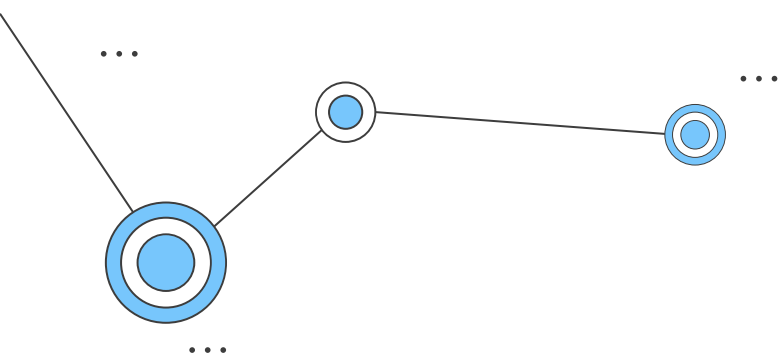
At some point, Alpha is marked as "stopped" and Beta as "live." Alpha sends a "reconnect" broadcast to all connected clients.

Client Reconnection

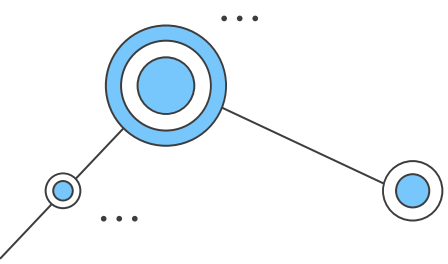
Upon receiving the "reconnect" signal, the client contacts the account server again.

Seamless Transition

The client connects to Beta without the player noticing any disruption, achieving zero downtime during the update.

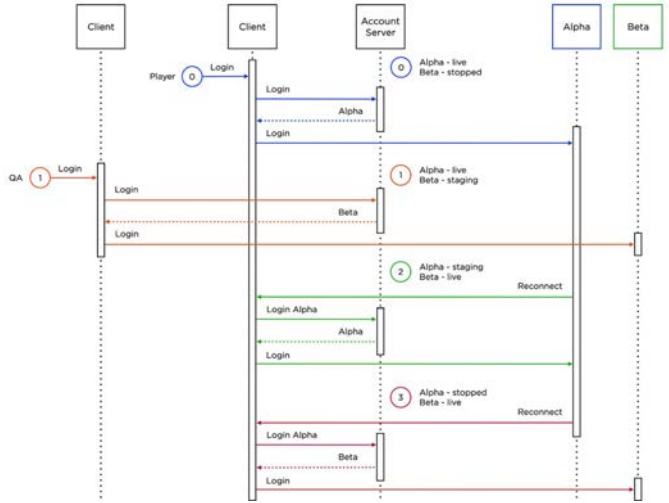


QA Testing and Client-Side Activity Completion



- **QA Final Test**: Introduce a "staging" status for the game server to allow QA specialists to run final tests before letting players join.
- **Client Activity Completion**: Aim to allow clients to complete certain activities (e.g., battles) on the same server they started on.
- **Staging Status Access**:
  - **QA Specialists**: Can access the game server during its "staging" status for testing.
  - **Ordinary Players**: Can also access, provided the client specifies the preferred game server in the login request.

Flexible Server Update Strategy



Initial Setup

Alpha is live, Beta is stopped, and the client is connected to Alpha.

Introduction of Staging

Alpha remains live, Beta turns to staging.

Staging to Live Transition

After QA checks, Beta becomes live, and Alpha turns to staging.

Alpha Stopped

Once Alpha is marked as "stopped," all new game access attempts are directed to Beta.

Version Management Strategy

Backward Compatibility

Rolling out a new game version initially necessitates maintaining backward compatibility for the client-server protocol.

Double Work Avoidance

Decided against maintaining both forward and backward compatibility to reduce workload.

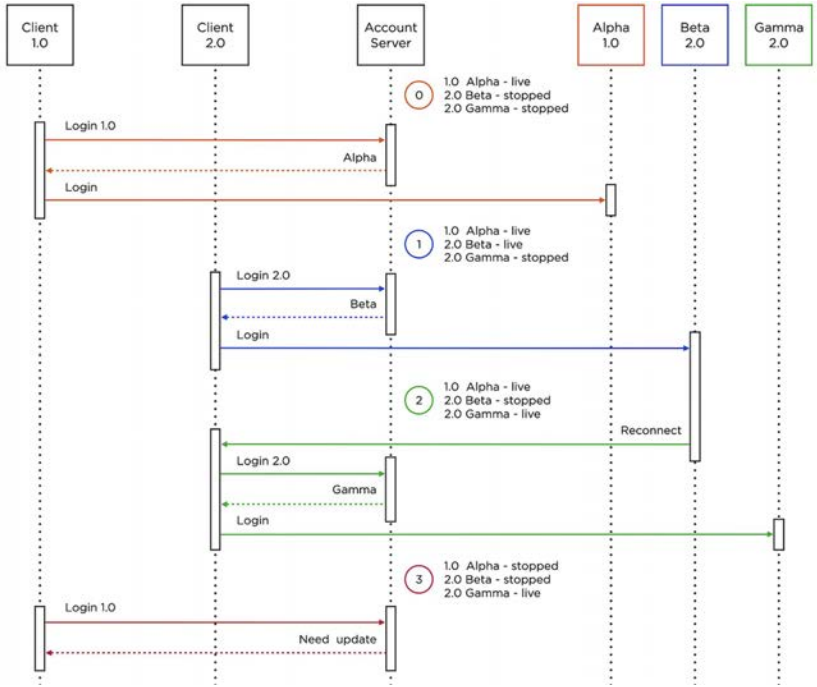
Strict Version Correspondence

Version X clients interact only with version X servers, and version Y clients with version Y servers.

Within-Version Flexibility

Changes to the server implementation are permitted as long as they don't affect the client-server protocol, eliminating the need for backward compatibility maintenance.

From Soft to Hard Updates



Release of New Version

At a certain point, version 2.0 is released to replace the existing 1.0 version.

Soft Update Activation

After QA checks, Beta becomes live for a limited percentage of players, offering the new 2.0 client. If successful, it becomes available to all players. No reconnects are sent from the old server version.

Error Handling

Any issues discovered during the soft update are fixed. Players are then transferred from Beta to a new server, Gamma, which incorporates these fixes, using the BGD process. Players on client 1.0 can still use Alpha.

Hard Update Activation

Eventually, Alpha is stopped, and all attempts to log in with the 1.0 client are blocked. Players are prompted to update their client to version 2.0.

Simplified Update via Game Tool

Statelessness of Account Server

The account server's entire state is stored in its own database, making instances stateless.

QA-Controlled BGD

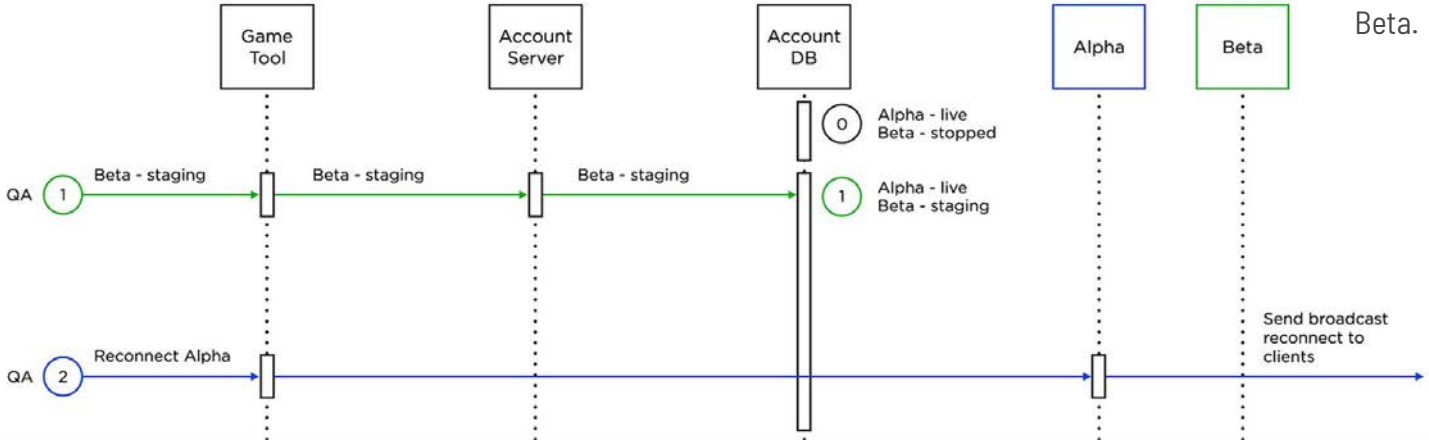
QA specialists can use the Game Tool to instruct the account server to change a game server's status.

Database State Update

Upon receiving the command from the Game Tool, the account server updates its database with the changed status of the game server.

Client Migration

After confirming that Beta is ready to go live, a QA specialist uses the Game Tool to instruct Alpha to send reconnect signals to clients, initiating their migration to Beta.



Bug Management

Zero Downtime for Fixes

The BGD strategy allows for bug fixes to be rolled out without causing any downtime, ensuring continuous gameplay for users.

Server Over Client

Bugs on the client-side are generally more dangerous due to longer update cycles, making the server-side BGD approach a valuable asset for maintaining game integrity.

Activity Continuity

In case of a critical error in a game activity, players can still participate in other game activities.

Client-Side Bug Mitigation

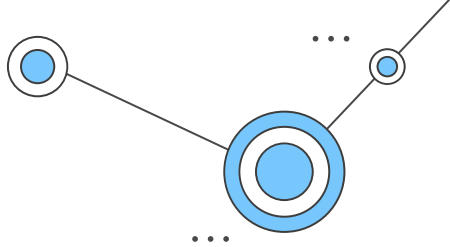
While updating the mobile client takes time, server adjustments can sometimes "persuade" the client to behave in a way that makes a bug invisible or non-existent.

Fallback Option

Even if preventative measures fail and bugs make it to the live environment, the BGD strategy provides a safety net for rapid remediation, sometimes in unexpected scenarios.

Conclusions

Summary and Takeaways



Robust CI/CD

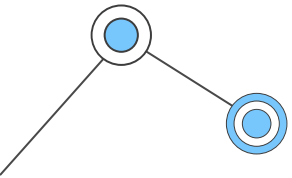
Serves as the backbone for integrating a large development team, enabling seamless integration and frequent updates without user disruption.

Flexible Backend Infrastructure

Meticulously designed for scalability to handle a growing user base without compromising performance.

BGD Strategy

Ensures zero downtime during software updates, providing a reliable and smooth user experience.



Thank you!

Dmitrii Ivashchenko

Lead Software Engineer

CONF42