MULTIPLAYER GAME REPORT

Create **games faster** and scale your **studio**

You're not just building a game.

You're building a business.

Advice and insights from

BespokeCI

SOLSTA

BEAMABLE

GAMEYE

BugSplat

What’s inside

Stay a game developer, not a software engineer

Start making **money** from your games

Game development is a dream job for many. But moving to full-time can seem daunting. In this report, we'll look at exactly how you can make that transition.

We'll be answering three big questions:

- How do you create an efficient game studio?
- What are the best practices for making a fun game?
- Which tools should you use to make your process more reliable?

This report isn't about monetization strategies or squeezing every last penny out of your players.

Creating profitable games is all about your process.

You want your team to be able to collaborate with each other easily. You want to be able to scale and grow. And you want to save time and money, so you can focus on actually making a good game.

How do successful studios run their business?

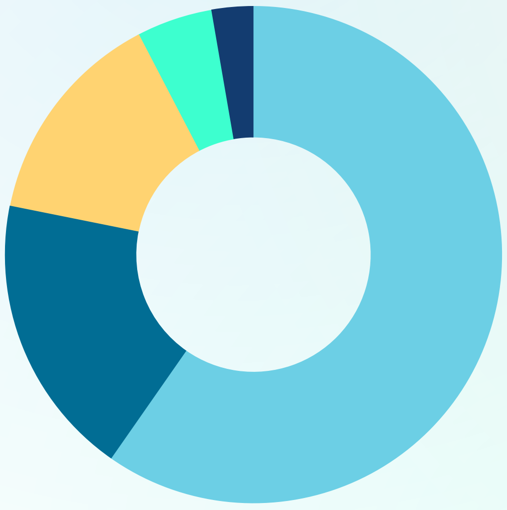
In today's age, it's never been easier to develop and release a game. There are so many tools and services that make the process far easier than it ever was before. But unfortunately, many developers don't earn as much as they'd like. **They create a great game but don't do well commercially.**

According to VGInsights, over half of indie developers on Steam made less than $1,000 from their games. In fact, it generally takes about three games to start making more than $100,000.

No' of devs by lifetime gross revenue

VGInsights: As of Feb 2022 (# of devs on Steam)

●	<$1K	**25.1K devs**
●	$1 – $10K	**8.8K devs**
●	$10K – $100K	**5.6K devs**
●	$100K – $1M	**3K devs**
●	$1M+	**1.6K devs**



No' of devs by lifetime gross revenue

VGInsights: As of Feb 2022 (# of devs on Steam)

	Avg. revenue per game	Avg. #1 of games released
The learner	$294	1.1
The hobbyist	$2,274	1.6
The indie	$14,999	2.4
The full-timer	$111,172	3.0
The success story	$3,655,808	4.5

These stats can be misleading at first glance. Most developers in the lower brackets are learners and hobbyists. It's so easy to make and release a game that these developers skew the data. We can ignore those.

You're far more likely to be in – or heading towards – the middle category: the serious game developer. You've made a nice amount from your games, and now looking to create or scale your studio.

How do you move along the **{brackets?}**

There’s a consistent theme among the top developers. They’ve been persistent and learned valuable lessons along the way. **They’ve started treating their studio like a business**.

So what exactly have they learned?

1. Playtest often and have a strong release strategy

You don't create a great game in a vacuum. You need to start getting feedback as soon as possible. As soon as someone else can play your game, you should be playtesting.

How else can you know if your game is fun? It's only when real people get their hands on your game that you can realise whether the core loop works or if a mechanic is balanced.

But you need to be careful when releasing early. Build too much hype and release with lots of bugs, then you're just going to disappoint your players. Better to start playtesting internally and then quietly release an early access version to iron out any big mistakes, before you make a big public splash.

2. Take ownership of more than just the game

It's not just about the gameplay and story. Obviously, those are essential. But you're also responsible for everything around the game, particularly if you're including multiplayer or any subscriptions. Now, you don't just have players. You have customers. Customers who'll get angry if your infrastructure fails and your game goes down.

In fact, it's not just your game going down that could be a problem. You also need to make sure that players have a fast and stable connection. Latency is the new downtime – too much lag is the same as the game going down.

So you need to make sure the game is fair and balanced. Match people of similar skills. Keep your servers online. Fix problems and crashes. Manage game updates while players are actually playing.

That's not to mention all the unglamorous parts of game development. Creating a game is only the beginning. You also need to package your build, distribute it to players, run marketing campaigns, and host the data somewhere. How do you do all that efficiently?

3. Be aware of the ongoing costs

The only way to keep on top of these responsibilities is to run your studio like a business.

And with that, there are three areas that will inevitably affect your costs: People, Processes and Products. The three Ps. They're the cornerstone for everything your business does. Every task is going to impact one of these three areas, so you want to keep the direct (and indirect) costs as low as possible. The trick is how.

4. Get help, so you can focus on your game

The best way to keep your costs down is to find the right balance between making or buying. Do you spend time making it yourself or use a third-party tool? Every project will be different, of course. But the areas you want to work on yourself should be where you're bringing the most value to your customers.

It might be worth buying temporarily, though. Just make sure you set a defined threshold for when you make the switch. For example, you might decide to use a third-party authentication system until you get a million players, at which point you can make your own.

It's all about managing **the flow of data** efficiently

Everything comes down to data. You need to check that your game actually builds and works correctly. You can test, but unfortunately the majority of errors will crop up in production. Once your game is live, there will be a ton of data flowing between you and your players, as well as between your players.

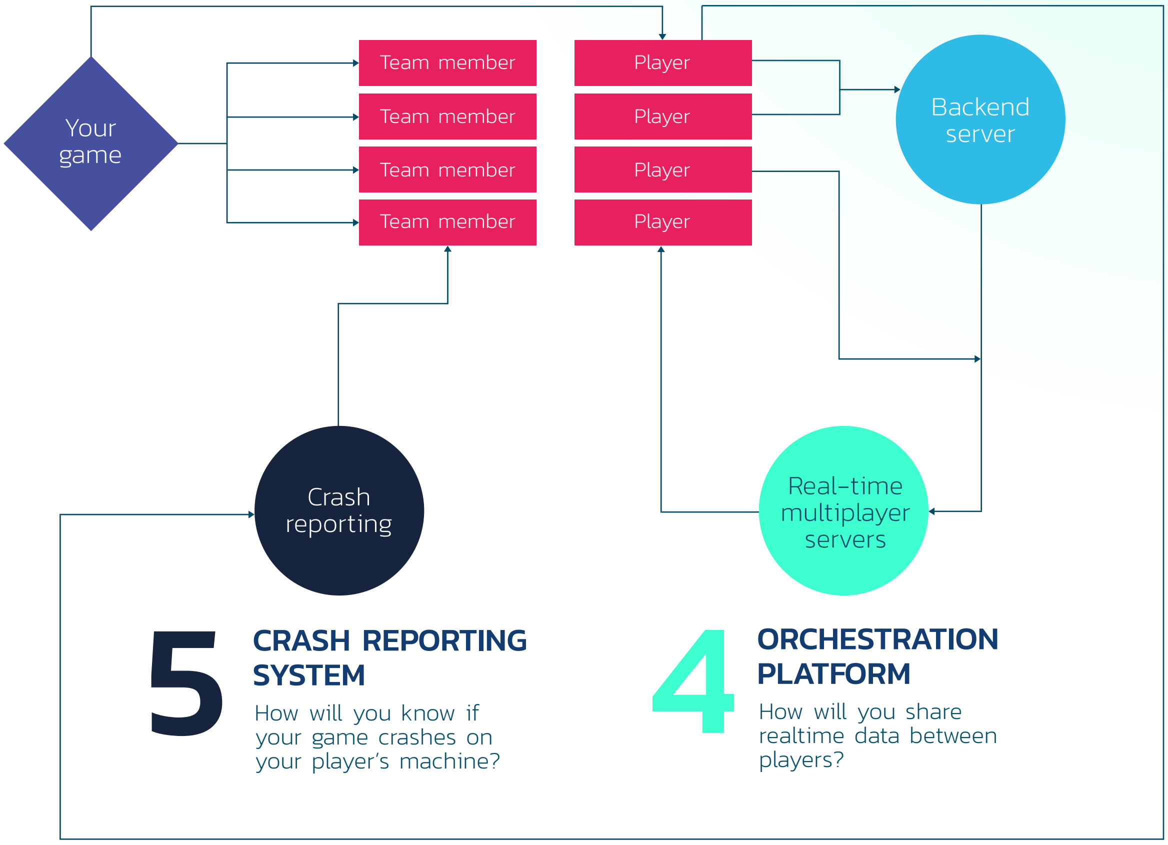
You want to be able to see and understand that data. So make sure you have the right observability tools in place (and the right alarms). Of course, monitoring alone isn't enough. You want to manage and tackle updates correctly, too. Sometimes, you might even need to run multiple versions of your game at once, before you decide to roll out globally.

How data flows through your game

1 BUILD SYSTEM
How do you turn your game into a playable file?

2 DISTRIBUTION SYSTEM
How do you get your game to your team members and players?

3 BACKEND SYSTEM
How do you store data that multiple people can access?



5 CRASH REPORTING SYSTEM
How will you know if your game crashes on your player's machine?

4 ORCHESTRATION PLATFORM
How will you share realtime data between players?

All of these tasks are about managing and moving around data. The faster you can do them, the faster you can playtest and improve your game.

Online games are large, complex programs that generate a lot of data. So there are five parts of your process that you need to make sure runs smoothly.

1. Your build system

How will you compile your data into a single file? The faster you can create a new build, the faster you can playtest. But you don't just need to create your final build, you need a way to make sure that it works.

2. Your distribution

How will you transfer large amounts of data? Your team needs to regularly download your game from somewhere. But you don't want to wait hours every time you release a new version.

3. Your backend

How will you store data that multiple people can access? As soon as you want to share data between two people – even a simple leaderboard – you need a central server to store it.

4. Your orchestration

How will you share real-time data between players? Online matches rely on fast connections, so you need to make sure you're hosting those sessions as close as possible to the players.

5. Your crash reporting

How do you collect data about problems? You can't predict how your game will react to every machine. But the only way you'll know if it's crashing is if you have a way to collect that data.

Stay a game developer, not a software engineer

All these tasks have one thing in common. They're the unglamorous parts of game development – the nitty-gritty reality behind the scenes. In fact, they're not game development at all. They're about software engineering or network management.

As essential as these elements are, they're not what differentiates your game from anyone else. It's possible to create these systems yourself, but it's not a productive use of your time. You wouldn't expect an accountant to create their own spreadsheet editor.

The difference in game development is that most developers don't realise the tools are already out there.

So we've spoken to the experts in each of these five areas to help you understand the best practices and what to look for in your tools.

Choose a build system

How do you playtest **faster?**

Focus on perfecting your build process

A game isn't a game until you've got that final executable that someone can double-click and install. It's the moment that everything comes together. But there's typically a whole process to taking all your changes and packaging them up into a form that people can actually play. This is the build process.

Running through that process manually can often take hours. And you can't be certain the end result will even run. The bigger your team gets – the more people working on the code – the more likely it is that someone will break the build.

"Some teams will have a ten-page document somewhere telling them how to create their build," Sean Saleh – co-founder and head of BespokeCI – told us.

"That document might have a list of tasks: install this library, change these settings to disable cheats, choose this server, download this certificate, update the version information. And that's just to do it on your own computer."

This is a tiresome process. At the very least, you want to create that document – you don't want that information only living inside one person's head. But ideally, you'll want to automate as much of it as possible.

Create a playtest build every day

New developers can often believe that creating their build is the final step. Finish the code, compile it, and simply send it off into the world. But the reality is that you'll probably want to create hundreds of builds before you're even close to launching.

"In the early stages of development, the most important thing you can do is play your game together," Sean explained. "But because building the game can be painful, developers often avoid doing it."

In fact – in an ideal world – you should be creating a new build every night. A fresh version, ready for your team to play together the next morning. The end of the day is a natural stopping point for everyone, making it an ideal cutoff point to make sure you're all working from the same version the next day.

Another reason to playtest regularly is to catch exactly which change caused a problem.

"Imagine you make a balance change to your game that makes the game less fun," Sean said. "How long does it take you to discover that? If you're playing every day, you find out instantly. If you're only testing every two weeks, it takes far too long to realise you made a mistake."

It isn't just about the quality of your game. It's also about getting into good habits once you've released your game into the wild.

"Building regularly lets you practise shipping something every night," Sean added. "You see some developers taking months to come out with their first patch or hotfix, because they're not familiar with making new versions. If they knew how to release changes more frequently, their players would be happier."

Build a clean version occasionally

One trap that developers can fall into is only ever using one machine. They don't create what's known as a "clean build" – a build on a fresh computer. Instead, they only ever create an "incremental build" – building on a machine that's done it before.

"Entropy can accumulate on a machine, though," Sean said. "So you try to install the game and it just won't work. For example, you might have deleted a library from your package, but that library is still on your computer. It'll run just fine on your machine, but not on anyone else's."

It's a tradeoff between time and reliability. You don't need to create a clean build every single time. A clean version can take hours, while an incremental one might only take minutes. So you want a mix of both.

Refine your process to iterate faster

No game is perfect after the first build. You'll inevitably find bugs, balancing issues and ways to improve the experience. It's only normal. There's a natural cycle to development: create a build, test it out, make changes, create a new build.

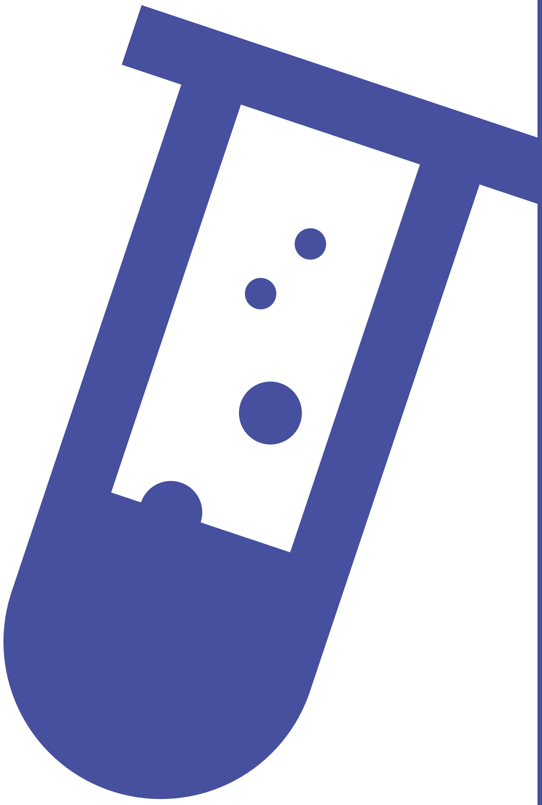
The trick is to make that cycle as smooth as possible. How can you go through the process as efficiently as possible? One way to check how well you're doing is to keep track of four key metrics:

- **Your lead time for changes.** How long does it take to make your changes?
- **Your deployment frequency.** How often are you making a new build?
- **Your change failure rate.** What percentage of builds fail?
- **Your time to restore service.** How fast can you get a working version out?

The first two metrics are subtly different from one another. You might be able to add new changes to your source code every hour. But you might want to wait until you've had a few come in before creating a new build.

"These metrics aren't just technical. They're organisational," Sean added. "You don't just fix a typo and immediately release it to all your players. You still have to deal with packaging it all up, getting it approved and releasing it across different platforms at the same time. So you probably want to just include it in your next patch."

There's another difference between your internal and external versions. Internally, you want to be regularly releasing new builds so your team can make sure it's fun. But externally, you need to make sure the build is stable. So your metrics for each type of build are going to be different.



Automate your process

The faster your development cycle, the more you can test it out. The more you test, the better your game will be. And the fewer mistakes you'll have, as you'll spot problems earlier.

Having a build system helps you streamline your process and playtest more often. BespokeCI is a build system that does all the heavy lifting for you.

- **Test before you commit.** BespokeCI not only automates your process, but checks whether your changes will break the build. Before it goes out to the rest of the team.
- **Create multiple builds.** Set up different builds based on what each team needs.
- **Integrate easily.** BespokeCI already has integrations with the most popular source control systems, like GitHub and Perforce, as well as Unity and Unreal.

"We walk you through the whole process, so you're only doing the bare minimum needed," Sean said. "You don't have to learn all the topics or hire a build engineer."

Get started early with BespokeCI

If you're early in your development process and haven't raised your funding yet, don't worry. **BespokeCI** offers help to indie developers to get them on their feet.

"We're happy to work with teams to get their build system set up early so that when they scale up their team, they have what they need," Sean explained.

Choose a distribution system

How do you collaborate as **a team?**

Transfer large files between team members

Games are inevitably large files. And whenever a player or a team member wants the latest version, they need to transfer that file and install it on their machine.

Not only do you want to do that as efficiently as possible, you also need to consider how you'll keep it secure, scale your storage, and deliver it to the end platforms.

Distribute multiple versions of your game at once

During development, you'll need more than one version of your game at any given time. Maybe you need to roll back to a previous version or you want to A/B test a new mechanic. Maybe you have separate teams working on different parts of the game and haven't brought those changes together yet.

"We've seen smaller indies use Steam, Unreal, or even Google Drive to store and distribute their builds," explained Fabian Ahmadi – the co-founder and head of Sales at Solsta.io

"But this is a hack. Those systems weren't designed around the complexities of the design process. Some studios might have 80 different environments they need to run. Steam just isn't intended to be used that way."

This creates a lot of bloat and repeated data to store all those different versions. And, inevitably, studios adjust their own process to fit around the systems they're using, rather than finding a system that works for them. But this can create bottlenecks that make it much harder to release on schedule.

Take a simple playtest. It's Tuesday night and you have a team of ten people. In the morning, everybody hits the server and tries to download the latest version. Even if you're not all in the same office, you're still going to get a congestion issue.

"You need a cache server – a repeater hub," Gammy Pichardo, Customer Success Manager at Solsta, said.

"You need that data to come over the local network, rather than the internet. But it's challenging to build something like that yourself – you'd have to manage a storage server on site that would be massive."

Distribute across different platforms

Creating a build for Mac, console or mobile takes a lot more effort than for PC. There's usually a manual process to download the files, make a few changes, and then install the game.

"Every platform has a different process," Fabian said. "What you need is a centralised control system – a single solution that can automatically grab the right build for the platform it's on."

Releasing a game can take even more steps, as many platforms have approval processes you need to go through. These steps can seriously delay a new patch on the platform and slow down your development cycle. The more you can reduce this friction, the easier it is to synchronise your releases so that your updates arrive to all your players at the same time.

Cut out the unnecessary risk

Manual steps always introduce an element of human error. The more you can automate, the less chance there is for a problem to occur.

"When people think about risk, they tend to think of big risks – the game failing or crashing," Fabian said. "But there are small risks, too. Six hours wasted time because the QA person downloaded the wrong version. That's risk."

These small risks are more insidious and they can compound. Each one has the potential to seriously delay your game or anger your players.

"Making games is already a risky job, so you want to eliminate as many as you can," Fabian added. "There's a lot you can't control. But your distribution is a risk that you don't need to have."

Upload only the changes

As you can see, transferring and storing large amounts of data is extremely time-consuming and difficult. That's why Solsta has a different approach. Rather than transferring the whole file, they split the files into more manageable chunks and only transfer the ones that have changed.

"We never upload the same piece of data twice," Gammy said. "We scan, transform, compress and secure your data and you upload the differences. This way, it doesn't grow exponentially and your storage footprint can stay small."

Not only does it make transferring files more efficient, but it gives you much more control over your data and helps you keep it secure.

- **Keep track of your version history.** Easily roll back to an older version without needing huge amounts of storage for backups.
- **Automate your process.** Solsta integrates with your build system so that you can easily make sure that you always have access to every version of your game.
- **Release across platforms simultaneously.** If you need to release on console and PC, you can easily distribute your builds to any device.
- **Keep your data secure.** Regardless of your size, it's important to know who has access to your game. Solsta is SOC2 and ISO 27001 certified and also supports Fine Grain Authentication to define your users and keep track of who has access.

Start using SOLSTA for free

Teams of up to five people can sign up and start using Solsta for free. So you can try it out and see whether it works for your studio.

Choose a backend system

How do you get to market **faster?**

Add features easily with a backend system

The moment you want to share data between two clients, you need a central place to store that data. Leaderboards, guild information, chat functions, news feeds, usernames, subscription information, and even seasonal event content.

That's your backend. All the content and information that isn't local to the player. The meta-features, mechanics and content that isn't baked into your game.

"In the old days, everything was stored on the device," Trapper Markelz – co-founder and chief operating officer at Beamable – told us.

"Nowadays, players expect an expansive and engaging metagame. But as soon as you need to share data between players, you need a backend."

Keep your game secure

Even just setting up a simple leaderboard can cause a headache. On the surface, it might seem like a trivial task. Just whip up a web app and set up an API.

But in reality, you're taking on a large responsibility. You're not just storing the score, you have the player's personal details. What happens when you have thousands or millions of players? Will your database that holds the leaderboard scale? And how do you make sure people don't hack your database, put themself at the top, and take the best prizes?

"The risks are high if you get it wrong," Trapper said. "You can ruin your company if the game can't stay online, if you can't scale to meet demand, if your game features don't perform, or if you get hacked."

Don't reinvent the wheel

Creating backend systems is a complex task. There are difficult technical challenges to overcome, and physical infrastructure to make it all work. While they're important to modern games, they're time-consuming to create yourself.

"The number one marker of a successful game is that it ships quickly," Trapper said. "Too many games only get to market when they're already running out of money. You want to get out early and start getting data from your players."

All the time and energy you spend creating backend systems yourself is time you're not working on the most important thing: making your game fun.

"Too many developers think they need to own everything. But these elements don't differentiate your game at all," Trapper explained. "Your inventory system is not what's going to set your game apart. The players won't care. So don't try to reinvent everything."

That doesn't mean you don't want flexibility. You should be looking for a backend that gives you a framework but doesn't lock you into any one way of doing things. For example, your matchmaker. Every studio is going to have its own criteria for what makes a good match, so your backend should allow you to create those rules yourself.

Keep your game alive

Games these days are going to have ongoing costs. Whether that's because you have some sort of progression system or multiplayer mechanics.

"Even small titles have rich in-game economies and systems of progression these days," Trapper said. "Now you're on the hook for paying and maintaining it. What if Apple changes its API so your backend needs to update? Your costs are going to continue."

While some costs will naturally scale with your players, you're going to have some amount of baseline. In fact, you should expect to spend about 10-15% on your backend server costs.

Treat everything as content

One way to keep your game alive is to release new content regularly. For that, you'll need a content management system. Thankfully, if you have a backend, you already have the systems in place to support that.

"What is a level in your game? It's a series of instructions of what to put on the screen," Trapper said. "Those instructions come from structured data. If you don't have a backend, that's all on the local device. The only way to change or add a level is to update the client."

Instead, you can have all this information stored in a content management system. The player can just pull the content when they load up the game. New offers in your store, new seasonal events, new character dialogue, new weapons, cards and powerups. It's all content.

Designing your game in this way allows you to continue updating your game, without needing to go back through approval processes or create and test a brand-new build.

Make cross-platform much easier

Cross-platform helps open your game up to more players. In fact, VentureBeat reported that it can increase revenue between 20% and 40%. So if you want your players to be able to play with people on other platforms or to take their data from PC to console, a backend helps you keep that data consistent between devices.

"It's all about the authentication," Trapper said. "Who is this person? Every platform has its own ID. If you only use, say, Steam, you're going to have trouble when they want to play on Xbox. But if you have somewhere to hold all those IDs, you can link them all to a single user."

This might not just be for cross-platform play. You might want to integrate with Discord or Twitch to give your players certain privileges or features. These integrations are much easier if you have that central place to store all the information.

Hit your launch deadline

The earlier you release your game, the sooner you can make improvements and start earning revenue. With a tool like Beamable, you can get all the backend features you need to get up and running quickly.

"We want to eliminate the need to have a backend developer in your team by providing you the workflow from inside the game engine itself," Trapper said. "Our goal is to give you the tools, so you can implement everything you need to realise your game's vision."

In fact, Beamable allows you to get all your backend features from a single SDK.

- **Run a wide range of backend tasks.** Manage your in-game economy, authenticate users, gather analytics data, and provide social features to your players using off-the-shelf managed services.
- **Customise game features for your needs.** With Beamable's C# Microservices architecture, you can easily edit the source code and add exactly what you need to your game.
- **Get the tools and data you need.** Access your game through a full-featured liveops portal for player CRM, analytics, and usage data.

Try out BEAMABLE for free

Developers can sign up and start using Beamable in just a few minutes. No need to talk to anyone. No need to pay anything. Just grab the SDK, install it and see if it works for you.

Choose your orchestration system

How do you keep your **players happy?**

Keep your multiplayer games running with an orchestration system

When you're running a multiplayer game, you need a server to host the multiplayer match. It's the only way to make sure that the match is fair and fun to all the players.

"In a competitive game, you can't have one of the players hosting the match. You want a trustworthy and reliable experience for your players," Roberto Sasso – chief technology officer here at Gameye – said.

"It just gives them an unfair advantage. It's not just that they could interfere with the match and cheat, they also have a time advantage. Their machine just knows what's happening before anybody else."

Milliseconds might not sound like much. But in a competitive first-person shooter, reaction times matter. A millisecond could be all you need to fire first.

The solution to these problems is to host the match on an independent server. That way, nobody has an advantage and nobody can cheat. But where exactly do you put that machine?

That's where orchestration comes in – deciding where to host your matches. And being able to take a different choice for each match.

Kill lag before it kills your game

Lag can ruin a player's experience. It breaks immersion and makes a game almost impossible to play. It's just frustrating.

"We've all had that match where you're jumping around the map, jittering back and forth," Roberto explained. "Finding the root cause might be harder than expected. In an online game, you have to deal with a multitude of mutating components because of factors you can't control."

There are two metrics that are important here. Bandwidth and latency. Bandwidth is how much data you can send at the same time. Latency is how long that data physically takes to get to its destination.

In real-time multiplayer matches, you're not usually sending huge amounts of data. But you do need that data to arrive as quickly as possible. So it's the latency that matters the most.

"The only realistic way to reduce latency is to host the match on a server as close to the players as physically possible," Roberto said. "You can't do anything about problems in the player's local network, but you can reduce the length of the trip. And, if you use your own matchmaker, you might have false positives on latency and bandwidth that's actually a bug in the matchmaking. So keep an eye on how your systems are interacting."

An orchestrator takes information about who should be in the match, then calculates the ideal server to host it.

Handle unpredictable spikes

There are plenty of reasons that your game might suddenly have a surge of new players. A streamer might have recently played it. You might be running a weekend offer. Rave reviews might hit the news.

But do you have the infrastructure in place to handle those new players? How do you scale to meet the sudden demand? If you can't, those new players can't find a match and get frustrated.

"Nobody really talks about how infrastructure can impact a game's success," Roberto said. "Cloud costs can easily rack up and eat into your profit margins if you're not careful. At the same time, cloud can help you tackle spikes. So you want to find the right balance between hosting types."

Balancing between all the possible solutions is time-consuming work. So you'll need to hunt for the right people to do the job.

You can see what happens when a developer gets this wrong. In 2024, Palworld ended up spending over $475,000 a month on server costs, according to PC Gamer. That kind of spending is just unsustainable in the long run.

"You really need to look at the cost per concurrent player before you launch a multiplayer game," Roberto said. "How much are you going to spend per player every month? And how are you going to cover that cost? There are ways to cut it down, but if you're planning on running a game for years – you're going to need a monetization strategy."

Test everything to avoid downtime

If your players can't play your game, they'll naturally get angry. So a single outage can cause a surge of bad reviews, which can seriously damage your game's reputation. For example, Helldivers 2's reviews on Steam dropped from "very positive" to "mixed" in just a few days following server issues, according to GameRant. This was primarily because they had only predicted getting 50,000 players, but ended up with hundreds of thousands.

It's impossible to predict exactly how many players you'll have. But, you can test different scenarios to make sure that your infrastructure can handle the load.

In fact, there are three main types of test you should perform:

- **Load** – Add more and more sessions and players to your infrastructure until you find the breaking point.
- **Soak** – Run your infrastructure for an extended period of time to see how long it can last before there are problems.
- **Spike** – Add a sudden surge of players to your infrastructure to discover what might cause it to break.

"Everything breaks eventually, so you need to know what causes those issues," Roberto explained. "If you know what the problems might be, you can preemptively solve them. For example, you might simply reboot your servers more often or add other ways to scale."

It's important to remember that testing your infrastructure is different to playtesting your game. You're not looking at whether the game is fun and balanced, you're looking for the hidden computing limits in your infrastructure. Computing limits that will destroy a player's experience and set your game up for failure.

Create a war room

One of the most difficult times for a new game is in the first few days after you launch or release a new event. In those moments, you're inevitably going to get a surge of new players. That's the whole point.

"These are the times when you need to have a close eye on your infrastructure," Roberto said. "You should have a war room, monitoring the health of your network and using observability tools to predict if there might be a problem. That way, you can jump on issues before a cascade effect impacts your entire infrastructure."

Whatever way you decide to keep on top of your infrastructure, the reality is the same: you need to be alert.

You need to treat your players like customers getting a service.

Automate your server infrastructure

Gameye's orchestrator automatically figures out where your servers should be and scales to meet the demand. All you need to do is tell the orchestrator which regions you want through the API.

"We can save developers up to 70% on their server costs, by using a mixture of bare-metal, cloud and edge computing," Roberto explained. "We also automatically scale across providers, so there's always a backup. Even if one provider has a problem, we can just switch you to another. Your players won't even notice."

The Gameye orchestrator was built from the ground up to deal with power-hungry games and has been battle-tested on multiplayer titles that demand a lot from their servers.

- **Get set up quickly.** All you need to do is pick your regions, containerise your game and upload the image. The whole process can be done in just an hour.
- **Reach players globally.** With servers across the world, you can easily handle players from Asia to America. **Pay only for what you use.** Gameye only charges you for the time you use, when you use it.

Playtest for free with GAMEYE

Gameye offers free servers to indie developers during their playtesting and development. So you can try it out and see if it works for you.

Choose your crash reporting system

How do you deal with **problems?**

Learn why your game is crashing with crash-reporting tools

No program is perfect all the time, on every machine. There are always going to be edge cases, surprises and exceptions. But you can't solve a problem if you don't know about it. So you need a crash reporting system in place to tell you when there's a problem.

"You spend so much time building your game, testing it, shipping it out, but you'll never know how it performs in the wild," Joey P – the product manager at BugSplat – told us. "Crash reporting is about figuring out what you need to fix, how stable your game is and whether you have any critical issues."

Video games are particularly vulnerable to crashes. They're complex programs that are tough on graphics cards and drivers, running on a huge variety of computers.

"You don't control what graphics card they're using, how much memory they have, or whether they've updated their drivers," Joey added. "It's impossible to test for every combination. So you need a way to identify what's causing the issue."

Find the root cause of problems

A crash reporting tool alerts you when a player's game crashes, but also tells you the line of code where the crash happened. It gives you an overview of the state the computer and game were in at the time of the crash. This makes it much easier to identify the root cause.

Without a crash reporting system, you're relying solely on what the player has told you. Players aren't professional testers, so their feedback can often be vague or unhelpful. Your QA team is now trying to replicate the problem with very limited information.

"If you have the right information, you don't have to replicate it," Joey explained. "Instead of chasing down the bug, you're able to see exactly what was going on when the computer crashed. Or you can prompt the player to give more useful feedback."

This drastically reduces the amount of time you're spending on fixing the problem. Your developers can see exactly which part of the code went wrong.

Prioritise what to fix

Not all crashes are equal. While crashes are extremely frustrating for the player, you don't want to waste your time on a problem that only one in a million people experience. You want to work on the most common crashes first. The best way to think about this is to consider how stable your game is. For how many players does it crash?

"About 10% of your defects are responsible for about 90% of your instability," Dave Plunkett – BugSplat's CTO and Founder – said. "But if you only rely on people sending messages, you're only going to hear from the loudest people. You need an automated way to track every instance of a crash and to group them together. Otherwise, you don't know which defects are the important ones."

Embed crash reporting early

Without these metrics, you can also end up with a false sense of security.

"About ten people will experience a crash before someone reports it," Dave added. "If you don't have a tool, you'll be surprised how often your game crashes without anybody talking about it."

You don't just want to add a crash reporting tool for when your game goes live. It's a helpful tool throughout the game development process.

"Crashes that make it to production are the hardest to fix as you need to ship a new version," Joey explained. "They're also the most insidious problems. It could be an error that's fundamental to your game, with loads of dependencies. The longer you leave that unfixed, the longer it takes."

BugSplat's crash reporting tool helps you not only get the crash data you need to solve the problem but also manage the process.

- **Get started with a couple of lines of code.** Adding BugSplat to your game is as easy as downloading the package and adding a line of initialization code at the beginning. And in the Unreal engine, it's even easier. You just need to update your configuration files.
- **Improve your player experience.** When your game crashes, you can customise the popup. If it's a crash you know about, you can even include a message that explains how to fix it. You'e now turned that negative moment into a positive one.
- **Keep track of your versions.** If you think you've solved a problem, but it comes back – the crash will automatically get flagged as in regression.

Get alerts for free with BugSplat

BugSplat **has a generous free plan** that allows you three users, a month of data, and records up to 15,000 crashes a month. So you can get started right away.

Get one month free
Use code 'BugsplatGameDevOneMonth' and get one month free on BugSplat's Team plan.

Bring everything together

Time is money. If you have an efficient studio – one where your team can easily work through the development process – you can iterate quickly and release updates to your game faster.

In the long run, you don't just save money – you can focus your time and effort on more important tasks: making your game fun and exciting.

Take your time deciding

Every game is different, and you should always look at all your options before settling on the tools you want to use. Don't dive into getting everything from a single provider or make sure that it's easy to switch if you do. You don't want to get trapped with a big provider that doesn't give you the flexibility you need.

That's why we've brought together the tools in this guide.

Commit to your game

If you need a hand, we're here to walk you through the process of getting set up with the tools you need and making sure everything works.

1. **They all easily plug into your game engine**
   Even if you're using an obscure game engine or have built your own, all the providers in this guide will work with you to make sure their system works with you.

2. **They scale with you**
   As your studio grows, you'll have more team members and more players. In both cases, you want tools that can handle that growth – without becoming expensive.

3. **They're the experts in their fields**
   It's always better to bring in a specialist than a generalist. Their sole focus is on developing their tools and making them the best they can be – updating them when there are changes in the industry and coming up with new features. They spend all their time on their tools, so you don't need to spend any.

4. **They all work together**
   Everything in this guide works together, so you can make sure that you've got all the tools you need to run your studio like a business.

Meet your experts

Sean Saleh
Co-Founder at BespokeCI
contact@gamebreaking.com
Sean has spent the past 6 years becoming an expert in Game Build Systems. Starting with going deep into many different games' build systems and rewriting them, and now he's heading up a team making a product that easily supports many different developers at once.

Fabian Ahmadi
Co-founder at SOLSTA
fabian@solsta.io
Fabian Ahmadi is a co-founder at Solsta.io with over 20 years of experience in the games industry. He leads Sales and Business Development, helping studios streamline build distribution workflows. Fabian is also the co-founder of ConSpire Seattle, a networking event for games professionals in the Greater Seattle Area.

Trapper Markelz
Co-Founder and COO at BEAMABLE
trapper@beamable.com
Trapper Markelz, currently COO at Beamable, has over 25 years of experience building teams, companies, and delivering consumer-facing software products. He has served as CEO of MeYou Health, co-founded GamerDNA and 360voice.com, and was CPO at Disruptor Beam.

Roberto Sasso CTO at GAMEYE
sales@gameye.com
Roberto Sasso is CTO of Gameye. He was the former Managing Director of Racing Controllers manufacturer Cube Controls. Lead the architecture for Yoox. Was the CTO for PizzaBo. And steered companies through huge billion-dollar acquisitions.

Joey P, Product Manager at BugSplat
joey@bugsplat.com
Joey P. is the Head of Product at BugSplat, a proudly bootstrapped and independent small startup, where he has been driving product innovation and business operations for nearly 10 years. In addition to leading strategy, Joey remains hands-on with web development and UI/UX design.

SELECT
START
SELECT
START